TURING

cocos2d-x 手机游戏开发

跨 iOS、Android 和沃 Phone 平台

徐松林 黄 猛 编著



人民邮电出版社
POSTS & TELECOM PRESS

图书在版编目（ＣＩＰ）数据

cocos2d-x手机游戏开发：跨iOS、Android和沃Phone平台 / 徐松林，黄猛编著. -- 北京 : 人民邮电出版社，2012.1
ISBN 978-7-115-26766-5

Ⅰ. ①c… Ⅱ. ①徐… ②黄… Ⅲ. ①移动电话机—游戏程序—程序设计 Ⅳ. ①TN929.53②TP311.5

中国版本图书馆CIP数据核字(2011)第231388号

内 容 提 要

本书共有12章，以跨平台的手机游戏开发为主线，围绕着cocos2d-x引擎，由浅入深地讲解了智能手机的游戏开发过程、跨平台游戏引擎的原理、跨平台游戏引擎周边工具以及跨平台的游戏开发，还给出了完整的实战案例，语言简洁，结构清晰。

本书适合所有想进入智能手机平台游戏领域的人士，包括在校大学生、游戏开发者和网络游戏公司的CTO等。

cocos2d-x手机游戏开发：跨iOS、Android和沃Phone平台

◆ 编　　著　徐松林　黄　猛
　责任编辑　明永玲
◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街14号
　邮编　100061　　电子邮件　315@ptpress.com.cn
　网址　http://www.ptpress.com.cn
　北京　　　　　　印刷
◆ 开本：800×1000　1/16
　印张：13
　字数：277千字　　　　2012年1月第1版
　印数：1－4 000册　　　2012年1月北京第1次印刷

ISBN 978-7-115-26766-5

定价：39.00元

读者服务热线：(010)51095186转604　印装质量热线：(010)67129223

反盗版热线：(010)67171154

序

近几年，开源力量在全球软件行业里扮演着越来越重要的角色。就移动互联领域而言，Linux内核被广泛用在嵌入式系统和无头设备，Android崛起，PhoneGap诞生，服务端LAMP已是标配，svn/git和Redmine成为很多软件公司的管理工具。时至今日，已经很难有哪家软件公司敢说他们完全不用开源产品了。国内的盛大和淘宝等公司都推出了开源平台，尝试开放企业的一些内部项目的源代码，这些都是非常有意义的举动。就国内开源历史来看，最成功的开源项目当属Discuz、PHPWind和Dvbbs，这3个开源项目服务了国内无数中小网站的站长。而cocos2d-x这一开源项目所追求的目标正是服务大量的手机游戏开发者，把大家都需要花费大量时间学习和需要花费时间重复做的工作抽取出来，形成一套开源的公用库。往大了说就是，希望能通过这套公用库来降低手机游戏开发的技术门槛和研发成本，使得中小团队和个人开发者能从中受益。

cocos2d-x仅是cocos2d社区的一个分支。cocos2d开源项目诞生于2008年第一季度，以阿根廷人Ricardo Quesada为社区领袖和主要开发者。最初这仅是一个用Python语言写的游戏框架，并没有体现出什么商用价值，但2008年正好是苹果发布iPhone 2.0 SDK的时候，这给全球的手机游戏开发者提供了一个非常好的舞台。cocos2d社区抓住了这个时机，在2008年就开出cocos2d-iphone分支，用Objective-C语言替换Python重写了整个框架。cocos2d随着iOS的发展而快速壮大。到了2009年的4月，用cocos2d-iphone写的游戏StickWars在App Store美国区的付费榜冲到第一，这个里程碑事件标志着cocos2d完全脱离了自娱自乐过家家的水平，进入了可以稳定商用的阶段。在2010年，cocos2d已经成为iOS平台上首选的2D游戏框架：几乎每个初学iOS游戏开发的程序员都会从cocos2d入手，社区里时常有开发者跳出来炫耀他的cocos2d游戏冲到App Store Top10，让人“羡慕嫉妒恨”。

cocos2d框架的最大优势在于其简单易学，游戏开发者只需潜心学习一个月左右就能掌握，不需要学习复杂的OpenGL ES知识就可以开发出一款商用水平的游戏，却能完全享受OpenGL ES硬件加速给游戏带来的性能提升和各种炫酷特效。这一核心特点成就了许多个人开发者和中小团队通过cocos2d快速在iOS平台上掘金的梦想。

前文提到，2008年cocos2d社区用Objective-C替换Python对整个游戏框架进行了重写，而在2010年当Android崛起到能够和iPhone抗衡，开发者希望能把游戏移植到Android平台上的时候，同样的事情再次上演：cocos2d-x分支被开出来，框架用C++重写。虽然C++广被诟病，但作为

iOS和Android两大平台都官方支持的编程语言，我们不得不“屈服”于C++的实力。事后证明这个技术路线的选择是对的！市面上各种Phone粉墨登场，都有自己独特的SDK、独特的UI控件接口，但对于游戏开发，它们具有两个相同点：一是支持C++开发，二是提供OpenGL ES接口。在这两点上，不论是国内的沃Phone、Ophone、乐Phone和魅族，还是国外的Bada和MeeGo等，都是一致的。甚至连封装层面甚高、直奔HTML5而去的WebOS都专门为游戏开发准备了C++&OpenGL ES专用的PDK。

和cocos2d-iphone的南美洲团队不同，cocos2d-x分支的主要维护团队在中国，我们有幸为大家做了这件事情，并很享受这个利他的过程。在大家的共同努力下，经过第一年的发展，社区里诞生了70多款基于cocos2d-x的游戏，其中包括多款Top 10的佳作，更有《捕鱼达人》、《Ah Up Planet》、《地铁总动员》和《91部落》等明星游戏。截至2011年10月中旬，基于cocos2d-x引擎创作的游戏在iOS平台已累计超过340万次下载，在Android平台则超过1000万次下载。

很快我们就发现大家都开始积极地回馈cocos2d-x项目。网龙公司为社区贡献了整套Lua绑定的代码，SuperRacccon、子龙山人、冬天的林和张文野等多位开发者不断撰写博客阐述使用要点，徐松林和黄猛更是热心地花费几个月的时间，为各位献上的这本教程。它不仅是cocos2d-x社区的第一本中文书，也是整个cocos2d社区的第一本中文书。

cocos2d-x的官方文档都受限于社区的国际化，因此没能提供中文版。另外，很多文档都是站在引擎开发者角度对原理进行阐述和解释的，对于初学者来说可能过于复杂。而此书从游戏开发者的角度，用通俗易懂的方式传授了cocos2d-x引擎的几个主要功能的用法，着重解决官方文档里没有涵盖的部分，如物理引擎和周边工具的使用，解答了新手经常感到困惑的热点问题，如开发环境安装、交叉编译和第三方库集成等。作者在这些地方都非常耐心地给出了按步骤的截图和代码段；最后，两位作者更是给出了一个很有分量的实际游戏案例，使得此书的学习价值大大提高。

非常感谢徐松林和黄猛两位作者的贡献，希望能有更多的游戏开发者从此书中受益。开源社区因为有大家的支持而更美好！

2011年10月

（本书封面素材使用了《捕鱼达人》iPhone版的游戏截图，特此感谢触控科技的慷慨支持。）

前　言

当前3G技术盛行，手机游戏已经得到快速发展。《植物大战僵尸》、《愤怒的小鸟》和《SNS的农场》等游戏先驱者已经在智能手机上大获丰收。

现在智能手机系统较多，作为手机游戏的开发者，我们需要同时了解多个系统的特性，并基于多个系统开发出自己的游戏，此时就需要解决跨平台的问题。好的跨平台引擎，可以做到编写一次代码，然后在多个系统上运行。目前cocos2d-x引擎已经能够支持微软的Windows系统、苹果的iOS系统、谷歌的Android系统以及中国联通的沃Phone系统，未来还将支持Bada和Symbian等系统。

本书面向的读者

本书由浅入深，从基础知识到引擎分析，再到实战游戏使用，结构清晰、语言简洁，非常适合想进入智能手机平台游戏领域的相关人士（大学在校生、游戏开发者、网络游戏公司的CTO）阅读参考。

本书的内容

本书以跨平台的手机游戏开发为主线，主要讲解了智能手机游戏的开发过程、跨平台游戏引擎的原理、跨平台游戏引擎周边工具以及跨平台的游戏开发，同时还给出了完整的实战案例。

本书中对开发手机游戏的相关知识（尤其是使用cocos2d-x引擎进行手机游戏开发）做了合理的划分，具体安排如下。

- 第1章是智能手机操作系统介绍，主要介绍几个常见的智能手机操作系统，包括苹果的iOS系统、谷歌的Android系统以及中国联通的沃Phone系统等。
- 第2章是游戏及游戏引擎介绍，主要介绍游戏的分类、游戏引擎的基础知识，同时简单介绍了游戏引擎中的渲染引擎和物理引擎等。
- 第3章开始进入cocos2d的世界，主要介绍cocos2d、cocos2d-iphone以及cocos2d-x引擎的基础知识，也给出了基于cocos2d-x引擎的优秀游戏分享。

- 第4章是搭建跨平台的开发环境，主要介绍如何在Windows系统上搭建跨平台游戏的开发环境，并以一个最简单的游戏为例给出其在各个平台上的运行效果。
- 第5章是cocos2d-x引擎的基础使用，主要围绕游戏开发过程中的各个环节展开介绍，包括游戏的整体架构、图形、动作和菜单等。
- 第6章是cocos2d-x引擎高级特性的介绍，主要介绍cocos2d-x引擎中的物理引擎、粒子系统和声音模块等。
- 第7章是cocos2d-x引擎的周边工具，主要介绍沃Phone应用程序打包工具、图片编辑器、地图编辑工具和粒子系统设计工具。
- 第8章是cocos2d-x引擎的交叉编译部分，主要介绍基于cocos2d-x引擎的游戏如何做到跨平台，同时以实际的游戏为例将其交叉编辑到iOS平台、Android平台和沃Phone平台上。
- 第9章是cocos2d-x引擎的实用篇，讲解了cocos2d-x引擎与游戏社交平台、手机广告平台和推广墙平台的集成方式，通过这一章的学习，我们可以充分利用社交模块为游戏添砖加瓦。
- 第10章和第11章讲述了一个完整的游戏案例的制作过程，这一部分会结合代码图文并茂地讲解游戏，从零开始讲解益智类RPG游戏（魔塔）是如何一步一步被设计和开发出来的。
- 第12章是未来展望，主要介绍智能手机系统和手机游戏的发展趋势，以及cocos2d-x引擎的发展方向等未来趋势。

本书的编写得到了王哲以及多位专家的大力协助，在此深表感谢。同时，在本书的编写过程中，我的妻子已经怀有宝宝，正是有了她的支持和体谅本书才能准时交稿。最后我希望所有阅读本书的人都能够从中受益，在手机游戏领域占有自己的一席之地。

徐松林

2011年9月

目　录

第1章

智能手机操作系统介绍

2007年1月9日，在MacWorld大会上，苹果首席执行官史蒂夫·乔布斯宣布iPhone产品正式推出，同年6月29日iPhone产品在美国上市。MacWorld是专门关注苹果公司产品的计算机杂志，也是全北美洲最畅销的关注苹果产品的杂志，每年的MacWorld大会，苹果公司基本上都会宣布一些新产品。

2007年11月5日，谷歌发布了基于Linux平台的开源手机操作系统Android。第一款基于Android操作系统的手机（T-Mobile G1）于2008年9月22日在美国纽约正式发布。

2008年，工业和信息化部、发改委、财政部联合发布《关于深化电信体制改革的通告》，开始电信业务重组：中国电信收购中国联通的CDMA网络，中国联通与中国网通合并，中国网通的基础电信业务并入中国联通，中国铁通并入中国移动。自此，国内电信运营商由5家变为3家。同年，中国移动、中国联通、中国电信分别获取国家的3G牌照：中国移动获得基于TD-SCDMA技术制式的3G牌照，中国电信获得基于CDMA2000技术制式的3G牌照，中国联通获得基于WCDMA技术制式的3G牌照。

2007年苹果的iPhone、谷歌的Android相继推出，2008年国内的3G牌照发放，一次又一次地向大家证明：3G的时代已经来临。

第三代移动通信技术（3G）是指支持高速数据传输的蜂窝移动通信技术。围绕3G技术的3G协议、3G手机和智能手机操作系统等，都涉及非常多的理论及相关知识。本书只介绍智能手机的操作系统以及智能手机操作系统上的应用和软件，至于其他（包括3G协议和3G手机）的理论和相关知识，请参考相关领域的相关书籍。

如果没有各类软件，电脑和手机就不能有序地按照我们的想法进行工作。例如，电脑上没有办公软件，我们就不可能按照自己的想法写出文章；电脑上没有网页浏览器，我们就不可能访问网页；而手机上没有收件箱，我们也就不能收发短信了。

如果没有各类游戏，电脑和手机也就不能满足我们在工作之余还想玩游戏的需求。例如，电脑上的魔兽争霸3（Dota）、地下城与勇士（DNF）等，手机上的连连看和愤怒的小鸟等。

那么，电脑和手机上的软件和游戏是怎么制作出来的呢？它们需要根据每台电脑和手机的硬件做特殊的处理吗？事实上，大部分的软件和游戏都是不需要针对机器做特殊定制的，因为它们都不是直接和机器硬件打交道，它们都不会直接操作机器的硬件，它们都是基于一个名叫“操作

系统”的程序开发的。

操作系统是一个管理硬件资源和软件资源的程序，不管在PC上还是在手机上，它都是核心和基础。操作系统管理全部的硬件资源和软件资源，控制软件的运行。

软件、游戏、操作系统以及硬件的关系如图1-1所示。

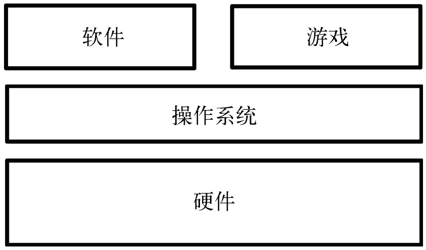


图1-1　软件、游戏、操作系统以及硬件的关系图

前面我们介绍了操作系统与软件、游戏之间的关系，现在来分别介绍iOS和Android等几个常见的手机操作系统。

1.1　iOS

2007年1月9日乔布斯宣布iPhone产品正式推出，同年6月29日，第一代的iPhone（iPhone 1）在美国发布。截至如今，苹果已经发布了4款iPhone产品，分别是2007年发布的iPhone 1，2008年11月发布的iPhone 2，2009年6月发布的iPhone 3，2010年6月发布的iPhone 4。

苹果公司修改了原来用于电脑的Mac OS操作系统，让其更符合移动设备，也就产生了iOS。苹果公司的iOS在4.0版本之前的名称为iPhone，在4.0版本之后才叫iOS。

iPhone是一个集照相手机、个人数码助理、音乐视频播放器为一体的掌上设备，在iOS 4.0操作系统上，iPhone自带以下应用程序：短信、日历、照片、相机、备忘录、YouTube、股市、地图、天气、实用工具文件夹、iTunes、浏览器Safari、邮件工具Mail、应用商城App Store以及系统设定等。下面我们开始介绍iOS的常见特性以及App Store应用商城。

1.1.1　iOS常见特性

iOS作为一款非常优秀的操作系统，是智能手机操作系统的典型。下面就给出其常见的一些特性。

(1) WiFi。支持无线网络（802.11b/g），通过无线网络能够方便地接入网络，手机还内置了浏览器Safari和邮件工具Mail。

(2) GPS。支持GPS定位。

(3) **摄像头**。iPhone 1和iPhone 2为200万像素，iPhone 3为320万像素，iPhone 4为500万像素。

(4) **闪光灯**。iPhone 4之前无闪光灯，iPhone 4支持LED闪光灯。

(5) **感应器**。也就是重力感应，iOS是第一个提供重力感应功能的手机操作系统。有了重力感应功能，手机能够根据用户水平或竖直的持有方式，自动地调整屏幕的显示方向。

(6) **光感器**。手机能够根据当前的光线强度，自动地调整屏幕的亮度。

(7) **超强的CPU**。iPhone 4采用1 GHz的A4处理器。

1.1.2 App Store

App Store是苹果公司为iPhone、iPod、iPad以及Mac创建的，它允许用户从中浏览下载为iPhone、iPod、iPad以及Mac开发的应用程序（包括软件和游戏）。有些应用程序是收费的，有些则是免费的。

App Store为第三方的开发者提供了一个方便、公平和高效的软件销售平台，正是因为这一点，手机软件业开始步入一个高速发展的时代。

App Store建立了一种三方（苹果公司、开发者和用户）共赢的商业模式。苹果公司掌握App Store的开发与管理权，是平台的主要掌控者，提供平台和SDK开发工具包，负责应用的营销，给开发者结算，向用户收费。开发者是应用软件的上传者，主要负责应用程序的开发，然后在App Store上自主运营（免费、收费、免费与收费之间调整、调整价格）自己的产品或应用。用户是应用程序的使用者，只需要注册、登录App Store并捆绑信用卡，就可以下载应用程序。

1.2 Android

2007年11月5日，谷歌宣布组建一个全球性的联盟组织，即开放手机联盟。开放手机联盟的成员包括手机制造商、手机芯片厂商和移动运营商。手机制造商包括摩托罗拉、三星和索尼爱立信等，手机芯片厂商包括美国的德州仪器、美国的NVIDIA和欧洲的ST等，移动运营商包括中国电信、中国移动、中国联通和德国T-Mobile等。

开放手机联盟将支持谷歌可能发布的手机操作系统或者应用软件，共同开发名为Android的开源的移动系统。

Android是基于Linux平台的开源手机操作系统，第一款基于Android操作系统的手机（T-Mobile G1）于2008年9月22日在美国纽约正式发布。

下面我们介绍Android的常见特性以及Android应用商城。

1.2.1 Android常见特性

这里给出Android的3个常见特性。

(1) **开放性**。Android平台是开源的、开放的，它允许任何移动终端厂商加入到Android联盟中来。由于它充分开放，越来越多的厂商推出基于Android的手机，Android的终端会越来越多。当然开发者也会越来越多，随着用户和应用的日益丰富，一个崭新的平台也将很快走向成熟。

(2) **丰富的硬件选择**。这一点其实还是与Android平台的开放性相关，由于Android的开放性，

众多的厂商会推出功能迥异又各具特色的多种产品。功能和特色的差异，却不会影响数据同步甚至软件兼容。

(3) **不受任何限制的开发商**。这一点其实与大家的利益息息相关。Android平台提供给第三方开发商一个十分宽泛、自由的环境，因此不会受到各种条条框框的限制，这样就会有非常多的新颖别致的软件和游戏诞生。但是这也有其两面性，血腥、暴力和色情方面的软件和游戏也会出现在Android里面。而且太自由宽泛的环境，很难建立一个像苹果App Store那样完整的产业链。

1.2.2　Android应用商城

尽管Android是一个开放的平台，但是谷歌还是建立了一个类似于App Store的应用商城，即Android Market（http://market.android.com/）。开发者可以上传自己的软件和游戏到Android Market上，然后在Android Market上运营管理自己的软件和游戏。

不过，由于Android太开放了，很多使用了Android系统的手机厂商都建立了自己的应用商城，例如摩托罗拉的智件园和HTC商城等。除了上述的几个商城外，常说的商城还包括亚马逊的Android商城。

1.3　其他手机操作系统

iOS和Android是目前最火的两大智能手机操作系统，但是除了这两大系统外，还存在着很多的智能手机操作系统，比如PalmOS、Windows Mobile、Windows Phone、塞班（Symbian）以及黑莓（Blackberry）等。中国移动的OPhone、中国联通的沃Phone和联想的LEOS是国内的三大智能手机系统，下面重点介绍一下。

1.3.1　OPhone

为了突破TD终端瓶颈，促进手机终端与中国移动的网络及应用服务进行无缝对接，中国移动和播思通讯在谷歌Android操作系统的基础上，主导开发了OPhone操作系统（Open Mobile Phone Operating System）。OPhone操作系统直接内置了中国移动的服务菜单、音乐随身听、手机导航、号簿管家、139邮箱、飞信、快讯和移动梦网等特色业务。

1.3.2　沃Phone

2011年2月28日，中国联通旗下自主的操作系统以及搭载该系统的多款机型正式发布。此次发布正式将操作系统命名为沃Phone操作平台（在此之前存在着多种命名方式，如Uphone等），这是中国首个自主知识产权的智能手机操作系统。

沃Phone是完全基于Linux内核的原生操作系统，具有自主、开放的特性。很难得的一点就是沃Phone是基于Linux内核的，而不是基于Android操作系统的，这与OPhone不同。

1.3.3 LEOS

2010年1月7日，在美国的CES 2010大展开幕之际，联想公司发布了一款全新的智能手机——乐Phone，其上市日期是2010年5月11日。

乐Phone采用的操作系统就是LEOS。LEOS是一个增强的Android操作系统，是联想根据自身的需求定制的Android操作系统。LEOS除拥有Android操作系统原来所有的功能外，还拥有联想自主开发的推送（push）引擎。推送引擎以联想公司强大的后台服务为基础，精准、实时地推送邮件、新闻、音乐、视频等信息，让用户在各种网络环境都能通过如影随形的推送服务实现互联之乐。

1.4 总结

本章一共介绍了5个智能手机操作系统，分别是iOS、Android、OPhone、沃Phone和LEOS。下面总结一下这些系统的整体特点。

苹果的iOS的产业链比较完善，对于开发者来说，只需要针对iOS系统进行开发，然后提交给苹果的App Store即可运营。

谷歌的Android、中国移动的OPhone、联想的LEOS都属于Android系列，Android的整个产业目前看起来是蒸蒸日上的，读者应该时刻都能从网上看到关于Android的新消息。但是由于其开放性，很多厂商对Android做了较大范围的改造，甚至已经宣称拥有自己的手机操作系统。这样开发者开发出一个软件或游戏时，要想做大规模的适配，有可能需要针对厂商的机型做定制。这些基于Android的厂商，都想像苹果那样控制整个产业链，每一家都在酝酿自己的应用商城，而且很多都已经上线，如谷歌的Android Market、中国移动MM商城、联想应用商店等。但是，任何单独一家Android厂商的终端数都小于苹果。因此，虽然看起来是一个繁荣的市场，但具体每个开发者从中如何获益，还得研究。

沃Phone是中国联通刚刚推出的智能手机系统，围绕该系统的终端还不是很多。不过，中国联通已经建立了针对该系统的官方商城、提供了专门的开发环境，而且由于该系统是中国联通自主知识产权建立的，所以拥有控制产业链的基础。

本章介绍的各智能手机操作系统的发展知识对读者来说比较重要，了解这些知识可以方便读者在进行软件和游戏开发时选择合适的操作系统。而在下一章，我们将介绍游戏以及游戏引擎的相关知识。

第 2 章

游戏及游戏引擎介绍

在本章中，我们首先介绍游戏以及常见的游戏类型，然后再介绍游戏引擎的架构。

2.1 游戏介绍

游戏的定义非常广泛，既可以指人的一种娱乐活动，也可以指活动过程。既可以是人手游戏、桌上游戏，也可以是电脑游戏。在本书中，游戏均指电脑游戏。

就目前来说，常见的游戏有以下一些类型：动作类游戏（ACT，Action Game）、格斗类游戏（FTG，Fighting Game）、射击类游戏（STG，Shoot Game）、第一人称视角的射击类游戏（FPS，First Person Shooting）、冒险类游戏（AVG，Adventure Game）、即时战略游戏（RTS，Real-Time Strategy Game）、角色扮演类游戏（RPG，Role Playing Game）。

从字面意思上，我们已经能够看出这些游戏类型的基本玩法，下面做进一步的阐述。

动作类游戏的代表作是《波斯王子》、《星球大战》等，这类游戏的主要玩法是玩家控制游戏内的人物角色用各种武器消灭敌人，目的就是为了过关，不追求故事情节。

格斗类游戏的代表作是《街头霸王》、《侍魂》、《拳皇》等，这类游戏的主要玩法是玩家控制游戏内的人物角色与另一人物角色进行格斗，另一角色可能是电脑内人工智能生成的，也可能是另一个玩家。

射击类游戏的代表作是《小蜜蜂》、《雷电》等，这类游戏的主要玩法是玩家控制游戏内的人物角色在一个卷轴式的游戏场景内进行射击游戏。

第一人称视角的射击类游戏的代表作是《CS》，主要玩法是玩家在游戏内的人物角色以第一人称的视角使用各种武器，主要是刀、枪等进行攻击，消灭敌人。

冒险类游戏的代表作是《神秘岛》等，主要玩法是玩家控制游戏内的人物角色进行虚拟冒险的游戏。故事情节往往是以完成一个任务或解开某些谜题的形式展开的，而且在游戏过程中着意强调谜题的重要性。

即时战略游戏的代表作是《魔兽争霸》，玩家在游戏内控制的人物角色为了取得战争的胜利，必须不停地进行操作，因为另一玩家控制的人物角色也在同时进行着类似的操作。

角色扮演类游戏的代表作是《最终幻想》、《仙剑奇侠传》等，这类游戏有自己完整的故事情

节，玩家扮演游戏中的一个或数个角色，比较强调剧情发展和玩家体验。角色扮演类游戏还可以进一步划分为下面几类：动作角色扮演类游戏（A-RPG，Action Role Playing Game）、策略角色扮演类游戏（S-RPG，Strategy Role Playing Game）、大型多人在线角色扮演类游戏（MMORPG，Massive Multiplayer Online Role Playing Game）。

除了上面这些传统电脑游戏外，这几年逐渐出现了一个新的游戏类型，这就是社交游戏（Social game）。社交游戏是一种运行在社交网站上，通过趣味性游戏方式增强人与人之间交流的网络应用。说起社交游戏，也许有人会感觉很陌生，但如果说到《偷菜》，可能就几乎无人不知、无人不晓了。在2009年，《种菜》、《偷菜》等在网上大行其道，而上网“偷菜”更是让很多网民“夜不能寐”。社交游戏在这一年获得了极大的发展。不管你是否接受、是否爱玩，社交游戏的确在影响着我们的生活，很多人在深夜两点起来玩游戏，只为了偷别人的菜。

社交网站中的社交游戏重视人与人之间的互动，经典的棋牌游戏在互联网上快速普及，在互联网发展初期已经形成了相当大的市场规模。社交游戏的成长加速了社交网站的规模化，虚拟礼品、虚拟宠物、投票等在社交网站上都得到快速的普及。现在，国外的Facebook和国内的人人网等社交网站上有着非常多的第三方应用。

2.2 游戏引擎介绍

上面介绍了很多种游戏，为了开发上述各类游戏，一般都需要引进一个或多个游戏引擎。

那么什么是引擎？顾名思义，引擎就是游戏的心脏。我们可以把游戏的引擎比作赛车的引擎。赛车的引擎，决定着赛车的性能和稳定性，赛车的速度、操纵感这些直接与车手相关的指标都是建立在引擎的基础上的。游戏也是如此，玩家所体验到的剧情、关卡、美工、音乐、操作等内容都是由游戏的引擎直接控制的。引擎扮演着总指挥的角色，把游戏中的所有元素捆绑在一起，指挥它们同时、有序地工作。简单地说，引擎就是用于控制所有游戏功能的主程序，从计算碰撞、物理系统和物体的相对位置，到接受玩家的输入，以及按照正确的音量输出声音等。

可见，引擎并不是什么特别的东西，无论是2D游戏还是3D游戏，无论是PC游戏还是手机游戏，无论是角色扮演游戏、即时策略游戏、冒险解谜游戏还是动作射击游戏，都要用到引擎，这个引擎或者是一个商用的产品，或者是一个开源的产品，或者只是一段起控制作用的代码。

经过不断地发展，如今的游戏引擎已经发展为一套由多个子模块共同构成的复杂系统，从渲染、动作到粒子特效，从物理引擎到碰撞检测，还有配套的周边工具，几乎涵盖了游戏制作的所有重要环节。在大型游戏中，DOOM引擎是2D、2.5D游戏引擎的代表，也是游戏引擎的鼻祖，而Quake引擎则是3D游戏引擎的代表。而在小型的游戏中，同样有着各类优秀的游戏引擎，cocos2d就是其中之一。cocos2d自身及其衍生的版本已经能覆盖C#、C++、Objective-C、Java、Python、Ruby、Lua、JavaScript等各种语言。图2-1是一个常见的小型游戏引擎的架构。

从图2-1我们可以看出，一个游戏引擎至少包括一个渲染引擎，另一部分物理引擎则不是必

需的。下面我们就介绍这两大部分。

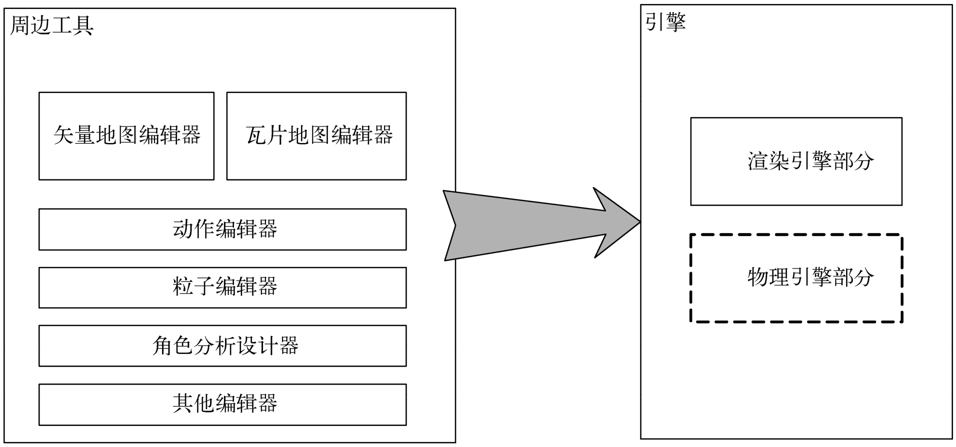


图2-1　游戏引擎的架构图

2.2.1　渲染引擎

渲染引擎是什么？渲染引擎的主要功能是让游戏元素可视化，让玩家可以看到游戏的logo、游戏的首页面、游戏的帮助和游戏的场景等，让玩家能够根据屏幕上看到的内容作出反应。

刚才提到的游戏logo、游戏首页面、游戏帮助和游戏场景是每一个游戏的基础组成部分，而苹果的iPhone，从一上市就带有非常完美而强悍的界面，当然iPhone的成功不仅因为有非常好的界面，但肯定也少不了界面的支持。上面这两点都足以说明，界面是一个产品的重要组成部分。为了显示界面，我们就需要渲染引擎。

当构造一个游戏引擎的时候，你通常想做的第一件事情就是建造渲染引擎。这是因为，如果看不见任何东西，你又如何知道你的游戏在运行呢?

最终用户对游戏视觉的要求也是很高的，一个游戏的美观界面绝对可以弥补其在其他方面的部分瑕疵。如果我们的游戏不能提供一个美观的界面，那么这个游戏绝对成为不了另一个“愤怒的小鸟”。

一般来说，渲染引擎的工作就是要创造出游戏的界面闪光点。要达到这个目的，不仅仅需要一个强大的渲染引擎，还需要大量的技巧，需要美工的鼎力支持，需要物理引擎千奇百怪的功能，需要图形编辑器和地图编辑器等的协助。正因为如此，渲染引擎才是在引擎当中最复杂的，它的强大与否直接决定着游戏的最终质量。

2.2.2　物理引擎

引擎的另一重要功能是提供真实的物理世界，这可以使游戏内物体的运动遵循真实自然的规律。例如，当角色跳起的时候，重力值将决定它能跳多高以及它下落的速度有多快，子弹的飞行

轨迹、车辆的颠簸方式也都是由物理系统决定的。

物理引擎通过为物体赋予真实的物理属性的方式来计算运动、旋转和碰撞反映。物理引擎使用对象属性（动量、扭矩或者弹性）来模拟物体行为，这不仅可以得到更加真实的结果，而且对于开发人员来说也比较容易掌握。好的物理引擎允许有复杂的机械装置，像球形关节、轮子、汽缸或者铰链。有些也支持非刚性体的物理属性，比如流体。

当然，也并不是所有的场景都一定需要物理引擎，有些简单的物理效果可以通过编程来实现。然而，当游戏需要比较复杂的物体碰撞、滚动、滑动或者弹跳的时候，通过编程的方法实现就比较困难了。比如说，在某个游戏中，游戏的主角是一架飞机，飞机需要在天空中飞行。虽然在飞行中不一定需要使用加速度或者减速度这样的效果，但是使用加速度或者减速度这样的效果会让游戏更像真实的世界。

我们再举个例子，在另外一个游戏中，有一处需要做到足球撞到墙体的效果。在没有物理引擎的时候，足球和墙体之间的碰撞由开发者写程序去判断，只能给出是否碰撞的结果，无法得到碰撞过程中足球和墙体在每一瞬间的状态。而使用了物理引擎，我们不仅可以知道足球和墙体是否碰撞，还可以知道碰撞前后的加速度变化如何，足球和墙体在碰撞前后各自的表面应该做什么反应，等等。通过物理引擎，实现这些物体之间相互作用的效果是比较简单的。图2-2就描述了上述的关系。

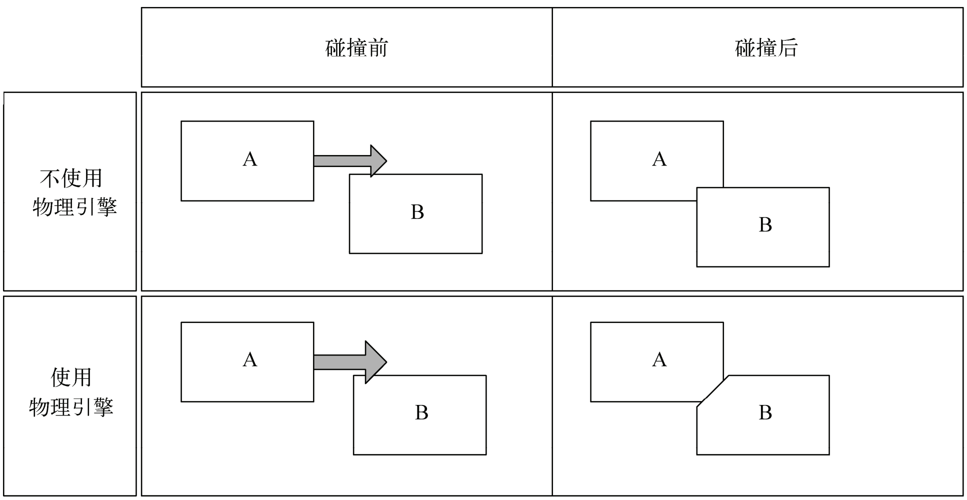


图2-2　使用物理引擎的效果

碰撞探测是物理系统的核心部分，它可以探测游戏中各物体的物理边缘。当两个物体撞在一起的时候，这种技术可以防止它们相互穿过，这就确保了当你撞在墙上的时候，不会穿墙而过，也不会把墙撞倒，因为碰撞探测会根据你和墙之间的特性确定两者的位置和相互的作用关系。

2.2.3 周边工具

游戏引擎除了需要常见的渲染引擎、物理引擎外，还需要很多的周边工具。这些工具包括角色分析设计器、瓦片地图编辑器、矢量地图编辑器、任务脚本编辑器、粒子编辑器、动作编辑器和图片编辑器等。

那么，为什么游戏引擎还需要这么多的周边工具呢？原因之一是方便开发者，之二是将游戏制作过程中的各个环节进行分割。有了周边工具，开发者在开发游戏的时候，就知道游戏的各个场景将使用场景编辑器来编辑，开发者只需要针对编辑器写通用的组件，然后有专门的策划人员去负责想场景。

下面我们来介绍常见的几个编辑器。

- **瓦片地图编辑器**。一个瓦片对应于一个矩形的区域，你可以把某块地方分割成3行3列，那就有9个瓦片，每个瓦片里面放上图片，不就是地图了吗？极端一点想，一个屏幕的分辨率是320×480像素，如果把一个像素就看成是一个瓦片，屏幕其实就是个320×480个瓦片组成的地图。因此瓦片地图类似于拼图游戏，又或者像搭积木。地图上的元素是可以重复利用的，元素之间的相互组合形成一个大的场景。瓦片地图编辑器就是一个为了方便进行瓦片地图制作的工具。
- **图片编辑器**。这种编辑器把散乱的图片保存成一张大贴图文件和一个图片坐标文件。为何要这样呢？主要是因为一般引擎在读模型文件的时候，读一张贴图比读*N*张贴图的运算量要小很多。
- **粒子编辑器**。这种编辑器通过图形界面配置粒子效果需要的参数。

2.3 总结

本章主要介绍了游戏以及用于制作游戏的游戏引擎。在游戏部分主要对现有的游戏进行了分类介绍，而游戏引擎则主要介绍渲染引擎、物理引擎以及周边工具。

接下来的一章，我们将进入cocos2d的世界，正是这个神奇的引擎，让我们编写跨平台的游戏成为了现实。

第3章

进入cocos2d的世界

大家都知道，手机已经成为我们最得力的帮手之一。手机用户的数量也将远远大于电脑用户的数量，我们在任何时间、任何地点都可以用手机来满足我们大部分的需求。而游戏是人的天性，游戏也将随着手机的发展得到进一步的发展。

无论你是喜欢苹果，还是喜欢Android，甚至是喜欢沃Phone，都没关系。这些智能手机系统都能够让任何个人或任何团体通过自己的努力，制作出精美的游戏，然后让这个精美的游戏得到社会的认可，包括获得回报。现在我们将介绍cocos2d这个引擎，它制作的游戏将覆盖前面提到的几个智能手机系统。

前面两章，已经介绍了手机操作系统、游戏以及游戏引擎的相关知识，现在将进入cocos2d部分。本章将分别介绍cocos2d、cocos2d-iphone和cocos2d-x相关的发展，并给出部分经典的游戏案例。

3.1 cocos2d 介绍

cocos2d最早是用Python（一种脚本语言）实现的一个开源游戏引擎，用来开发2D游戏和基于2D图形的任何应用。官方网站是http://www.cocos2d.org。

cocos2d的主要作者是Ricardo Quesada（他在社区里自己简称Riq），阿根廷人。cocos2d第一次发布的时间是2008年2月29日，版本号为0.1.0。从0.1.0版本到目前的0.4.0版本，每个版本的具体发布时间参见表3-1。目前cocos2d引擎正在开发0.5.0版本，暂未发布。

表3-1 cocos2d各版本发布时间

cocos2d版本号	发布日期
0.1.0	2008年2月29日
0.1.1	2008年3月10日
0.1.2	2008年3月18日
0.2.0	2008年3月20日
0.3.0-beta0	2008年5月5日
0.3.0-beta1	2008年6月2日
0.3.0-rc0	2008年6月27日

（续）

cocos2d版本号	发布日期
0.3.0	2008年9月5日
0.4.0-rc0	2010年7月7日
0.4.0	2010年9月8日

根据官方网站的描述，cocos2d的主要功能包括下列9个：

(1) 非常容易地进行不同场景之间的流程控制（Flow Control）管理；

(2) 菜单（Menus）维护；

(3) 支持标签和HTML标签的文本渲染（Text Rendering）；

(4) 快速而方便的精灵（Sprites）；

(5) 按照你的想法指挥精灵们变化，可组合的动作（Actions）包括移动、旋转和缩放等；

(6) 包括波浪、旋转和透镜等特效（Effects）；

(7) 支持矩形和六边形的瓦片地图（Tiled Maps）；

(8) 支持场景切换的特效（Transitions）；

(9) 支持硬件加速（OpenGL）。

cocos2d采用的开源协议是BSD协议，开发者可以放心地使用，而不必关心授权问题。官方网站还提供了大量的测试例子，当然也少不了编程指南、API参考和视频教学。当然，基于cocos2d引擎已经涌现出很多优秀的益智类游戏。

当iPhone和iPad在世界范围内疯狂时，cocos2d的作者也顺势推出了cocos2d的iPhone版本，这个版本就是cocos2d-iphone。

虽然cocos2d是所有cocos2d引擎家族的鼻祖，但现在人们提到cocos2d的第一反应已经是cocos2d-iphone了。cocos2d网站上说0.4.0版本发布2个月后达到了1700次下载，但同样的下载量，cocos2d-iphone版本只发布一两天就达到了，而cocos2d-x版本只发布三四周就达到了。

3.2　cocos2d-iphone 介绍

cocos2d-iphone是基于cocos2d的思想进行开发的，第一版本于2008年6月25日发布，官方网站是http://www.cocos2d-iphone.org/。由于iOS系统不允许开发者使用动态链接库的方式开发应用，所以Riq将cocos2d-iphone的开源协议从BSD协议改为MIT协议。

笔者认为Riq对cocos2d-iphone这个名字现在已经有点心有戚戚焉了，因为这个引擎现在不仅支持iPhone，而且还支持了iPad和Mac OS X。cocos2d-iphone中的“-iphone”已经不能正确表达该引擎的项目范围了。

cocos2d-iphone和苹果iOS一起成长起来，凭借着使用Objective-C语言和SDK无缝衔接的优势，获得广大果粉开发者的追捧和喜爱，现在已成为苹果iOS上2D游戏引擎的首选。

目前苹果App Store里有无数基于cocos2d-iphone引擎的游戏。提交到cocos2d-iphone官方网站登记的游戏数量已经接近2000个，并且以每天2～3个的速度递增，而实际游戏数量应该远高于此，作者个人估计至少是5倍或10倍于该数据。为了应对日益发展的iOS生态链以及cocos2d-iphone自身引擎的发展需要，cocos2d-iphone引擎的版本变更非常频繁。从2008年6月25日发布的0.1.0版本到2010年12月16日发布的0.99.5版本，中间发布了若干版本，据不完全统计有60个之多。

cocos2d-iphone在每个版本都给出一些令人兴奋的新特性，在0.90版本时进行了一次非常重大的升级，使用CC作为名字空间，自此cocos2d-iphone引擎开始使用这个全新的体系，也正是这些变动让其越来越完善。

3.3　cocos2d-x 介绍

cocos2d-iphone非常成功，围绕着cocos2d-iphone的各类开源项目越来越多，也越来越完善，cocos2d-x就是其中的一个典型代表。cocos2d-x的主旨是跨平台，所以不再采用苹果独家的Objective-C，而是采用更流行的C++。cocos2d-x的官方网站是http://www.cocos2d-x.org，采用的开源协议同样为MIT协议。

cocos2d-x目前已成为cocos2d官方的一部分了。cocos2d-iphone的主要开发者Riq已经同意将cocos2d-x的代码与cocos2d-iphone的代码进行同一个代码仓库管理，而且在后续CocosBuilder等动画编辑器的设计中也同时考虑支持cocos2d-x。

cocos2d-x中的 “x”有两个意思，一是表示“C++”，有时候，C++文件的后缀名可以用cxx代替，因此该处的x表示该开源项目使用C++编写，并且提供C++的API；二是表示“Cross”，该开源项目的目标是跨更多的平台。现在基于cocos2d-x引擎开发的游戏已经可以成功运行在很多平台上，包括iPhone和iPad 4.0以上版本、Android 2.0以上版本（包括3.0的Moto Xoom和三星GalaxyTab10.1等）、Windows XP、Windows 7、联通沃Phone、联想乐Phone和魅族M9。

cocos2d-x各版本的发布时间见表3-2。

表3-2　cocos2d-x各版本的发布时间

cocos2d版本号	发布时间
0.7.1	2010年12月20日
0.7.2	2011年1月26日
0.8.0	2011年3月15日
0.8.1	2011年3月21日
0.8.2	2011年4月8日

3.4　cocos2d-x 游戏分享

前面给大家简单介绍了cocos2d-x引擎，现在大家可能会问这样一个问题：目前有基于

cocos2d-x引擎开发并且发布的游戏吗？因为从发布出来的游戏数量和这些游戏质量可以看出cocos2d-x引擎的生命力。

cocos2d-x引擎开放的代码除了引擎代码外，还包括了额外两个项目代码，分别是Hello World和tests。其中tests是cocos2d-x引擎的单元测试项目，使用了cocos2d-x引擎的绝大部分接口，因此tests也可以作为学习引擎接口的一个方法。

除了cocos2d-x本身提供的这两个项目外，市面上已经出现了很多基于cocosd2d-x引擎开发的游戏。例如触控科技开发的《捕鱼达人》，MT工作室开发的《魔域之城》以及网龙公司开发的《91部落》。

3.5　其他 cocos2d 版本介绍

除了cocos2d-iphone、cocos2d-x外，还有很多其他的cocos2d实现，不过大部分都是基于cocos2d-iphone的代码进行其他的重新实现，下面列举其中的一些实现。

- cocos2d-net。cocos2d的.NET实现，运行在Mono上面。对于这个引擎，笔者很奇怪为什么选择了Mono，而不是Windows Phone 7呢？如果选择Windows Phone 7，这个引擎就会搭上Windows Phone 7这艘新船。
- cocos2d-windows。cocos2d的Windows实现，官方网站为http://code.google.com/p/cocos2d-windows/，使用GPL v3协议发布。该分支是一位韩国人所做，但和多数开源项目一样，在一次较大范围的提交（Release）之后就再无消息，没有留下文档，也没有更新。
- cocos3d。cocos2d的3D实现，官方网站为http://www.cocos3d.org。这是一个非常给力的分支，由加拿大人Bill Hollings开发维护。cocos3d不是一个完全的3D引擎，而是在cocos2d的CCLayer（层）上面扩展出3D世界，以cocos2d-iphone扩展包的形式运行。如果能够成功实现并商业化应用，那么我们就可以用cocos2d-iphone和cocos3d搭配做出2.5D的游戏。
- cocos2d-android。cocos2d-android是cocos2d的Java实现，并且能够在Android上运行，官方网站为http://code.google.com/p/cocos2d-android/，使用BSD协议发布。
- cocos2d-android-1。该分支是由国内资深开发者周为宽建立发展的。周为宽认为cocos2d-android的发展太慢，所以创建了一个新的Android实现，以实现最新的cocos2d-iphone版本，官方网站为http://code.google.com/p/cocos2d-android-1/，使用BSD协议发布。不过最近周为宽好像也不怎么维护这个分支了，该分支目前主要由俄罗斯人OpenGenius维护。OpenGenius直接翻译过来就是“开源天才”，多么霸气外露的ID啊。
- cocos2d-javascript。cocos2d的JavaScript实现，官方网站为http://cocos2d-javascript.org/，由一位新西兰程序员建立并维护。cocos2d-javascript把cocos2d整套框架体系移到了HTML5和JavaScript上，走到了时代前沿。
- LuaCocos2D。把cocos2d-iphone绑了Lua脚本，官方网站为http://github.com/boriscosic/

LuaCocos2D/，作者号称绑了Lua后成功把游戏发布到苹果的App Store上面了。可是这个项目开源出来的部分比较小气，只演示了绑CCMenu的一小段代码。

- ShinyCocos。把cocos2d-iphone绑了Ruby的实现，官方网站为http://github.com/funkaster/shinycocos/。

3.6 总结

本章介绍了cocos2d的发展历史，包括cocos2d、cocos2d-iphone和cocos2d-x的相关知识。至此，我们了解了操作系统、游戏引擎和cocos2d的相关知识。在接下来的章节，我们就开始使用cocos2d-x引擎开发游戏。

第4章

搭建跨平台的开发环境

通过前面几章内容的学习，各位读者应该已经掌握了智能手机系统、游戏、游戏引擎和cocos2d的相关知识。现在我们将正式进入游戏的实战部分。本章将首先向大家介绍跨平台的开发环境搭建，并且在各个平台上运行经典的Hello World。

4.1 环境说明

随着手机的相关技术越来越受到开发者的关注，各类手机的开发环境也逐渐丰富起来。手机平台阵营之一的苹果公司在iPhone和iPad发布时就提供了完整的解决方案：在Mac操作系统上通过Xcode、iPhone的SDK、iPad的SDK来编写iPhone、iPad的应用和游戏，而另一阵营的Android则可以使用Eclipse、Android的SDK、Android的NDK以及各个厂商自己的一些类库来编写Android的应用。其他的手机平台各自都有自己配套的开发环境。那么有没有一个开发平台可以覆盖这两大平台呢？答案是：有的。现在我们就开始介绍开发环境搭建的相关知识。

所谓“工欲善其事，必先利其器”，我们需要学会如何搭建支持开发跨平台游戏的开发环境。在前面已经介绍了cocos2d-x引擎支持开发跨平台的游戏，我们也选择了此引擎，因此，我们这里所说的搭建跨平台的开发环境，也就是搭建cocos2d-x的开发环境。

一个完整的环境包括开发、编译、链接、运行和调试等几部分。如果这几个部分全部手动来处理，那我们的很多时间将花费在与游戏无关的内容上面，这样很容易得不偿失。所以我们选择了微软的Visual Studio作为我们C++的集成开发环境，在Visual Studio里面，我们可以开发游戏，可以通过Visual Studio进行调试，还可以编译链接成可运行在Windows上的版本。

可在Visual Studio里面，我们只能编译出Windows的版本，但是我们的目标不仅仅是Windows版本，我们还需要支持iOS、Android和沃Phone版本，因此我们需要安装这几个平台自己的软件开发包（SDK）。有了这个SDK之后，我们才能够编译出在各平台上运行的软件包。

下面我们就开始一步一步地搭建跨平台的开发环境。

4.2 环境搭建

我们现在开始搭建整个环境。我们需要安装的软件包括集成开发环境（Visual Studio）、交叉

编译工具（Cygwin）以及平台支持（iOS、Android和沃Phone）。

4.2.1 安装Visual Studio

Visual Studio是微软公司推出的集成开发环境，它可以用来创建 Windows 平台下的 Windows 应用程序和网络应用程序，也可以用来创建网络服务、智能设备应用程序和 Office 插件等。Visual Studio是目前最流行的 Windows 平台应用程序开发环境，也是目前Windows平台上最流行的C++开发环境，目前已经开发到 10.0 版本，也就是 Visual Studio 2010。

cocos2d-x支持的Visual Studio版本包括Visual Studio 2008、Visual Studio 2008 Express和Visual Studio 2010。本书选择的Visual Studio版本为Visual Studio 2008的90天试用版，Visual Studio 2008是运行cocos2d-x最稳定的版本。具体的安装步骤如下。

(1) 首先从微软公司官方网站下载Visual Studio 2008，没有购买的话有一段时间的试用期。如果既不想购买，也不想有试用期，可以下载Visual Studio 2008 Express。

(2) 双击Visual Studio 2008的安装程序，安装系统会开始加载文件，加载完成后就可以进入安装界面，如图4-1所示。



图4-1 Visual Studio 2008的安装界面

(3) 单击“下一步”按钮，Visual Studio 2008安装程序会检测当前机器是否安装了Visual Studio 2008的部分组件，然后给出已经安装的Visual Studio 2008组件列表和没有安装的Visual Studio 2008列表，如图4-2所示。在这一步同时会有一个Microsoft软件许可条款，必须阅读并接受这个条款安装程序才能进行下一步。

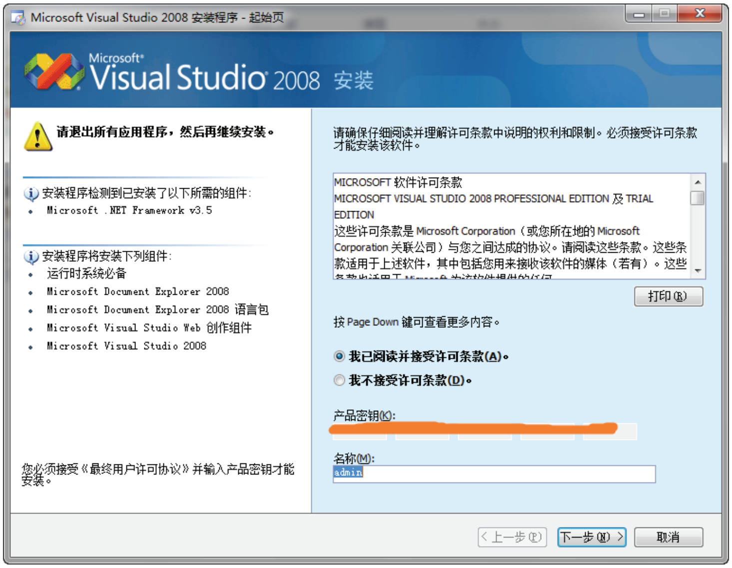


图4-2　Visual Studio 2008安装程序的起始页

(4) 单击“下一步”按钮，Visual Studio 2008会给出“默认值”、“完全”和“自定义”三个选项，如图4-3所示。如果对Visual Studio不是很熟悉，我们建议选择“默认值”。如果对Visual Studio很熟悉，那么就请按照自己的需要选择对应的组件。在这一步，我们还需要设置Visual Studio的安装路径。

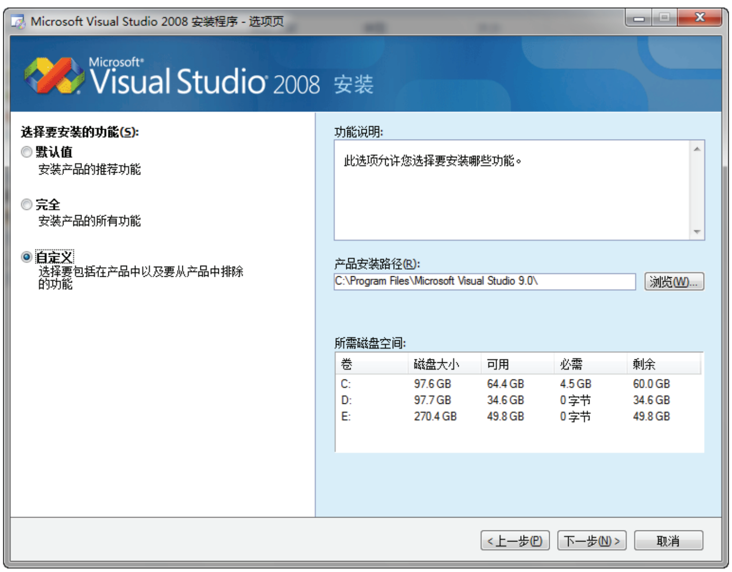


图4-3　Visual Studio 2008安装程序的选项页

(5) 接下来的几步比较简单，直接单击“下一步”按钮即可，出现图4-4则表示Visual Studio已经安装成功，然后单击“完成”即可结束安装过程。



图4-4　Visual Studio 2008安装程序的完成页

(6) 在Visual Studio安装完之后，可以根据自身的需要选择是否安装Visual Studio 2008的SP1和MSDN Library For Visual Studio 2008，下面分别介绍一下这两个组件。

和Windows系统一样，Visual Studio 2008在正式发布以后，同样会出现一些补丁，当这些补丁积攒到一定程度后，为了方便使用者，微软公司会发布该产品的SP补丁包。Visual Studio 2008的SP1就是Visual Studio 2008的第一个SP补丁包。

MSDN 的全称是 Microsoft Developer Network，它是微软公司面向开发者的一种信息服务。MSDN 实际上是一个微软产品的开发虚拟社区，包括技术文档、在线电子教程、网络虚拟实验室、微软产品下载（几乎全部的操作系统、服务器程序、应用程序和开发程序的正式版和测试版，还包括各种驱动程序开发包和软件开发包）、Blog、BBS和MSDN杂志等一系列服务。当然大家常说的MSDN实际上是MSDN Library。MSDN Library涵盖了微软全套可开发产品线的技术开发文档和科技文献（部分包括源代码），当然也包括一些MSDN杂志节选和部分经典书籍的节选。与Visual Studio 2008配套的MSDN Library是MSDN Library For Visual Studio 2008。

4.2.2　安装Cygwin

Cygwin是一个在Windows平台上运行的Linux模拟环境。它对于学习Linux操作环境，或者从Linux到Windows的应用程序移植，或者进行某些特殊的开发工作，尤其是使用GNU工具集在Windows上进行嵌入式系统开发，非常有用。

本书选择的是Cygwin的本地安装版，安装程序的版本号为2.721。下面介绍一下具体的下载和安装步骤。

(1) 首先下载Cygwin，Cygwin的安装文件很容易在网络上找到。目前国内的网站上有“网络安装版”和“本地安装版”两种。标准的发行版应该是“网络安装版”。不过由于Cygwin安装时需要下载的文件地址大部分都在国外，因此对于网速不好的朋友，建议使用“本地安装版”。但是两者的安装并无大不同。

(2) 双击安装文件Setup.exe，安装程序会进入安装界面，如图4-5所示。

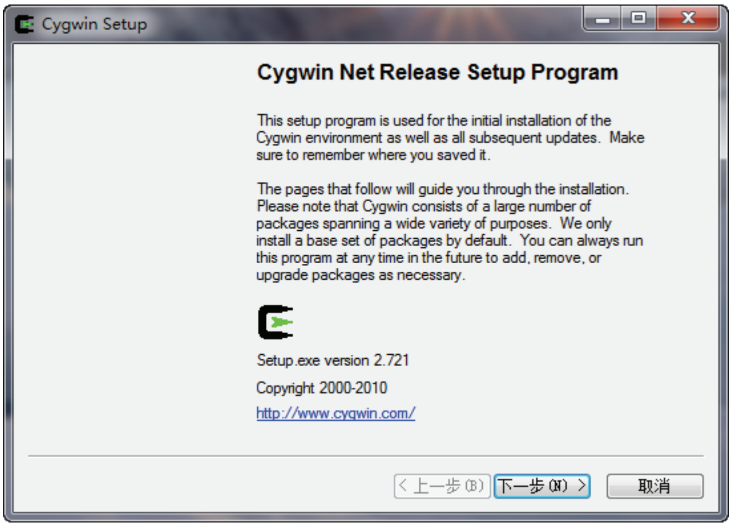


图4-5　Cygwin的安装界面

(3) 单击“下一步”按钮。安装程序会给出安装方式的提示，一共有三种，分别是“Install from Internet”、“Download Without Installing”、“Install from Local Directory”，如图4-6所示。“Install from Internet”是直接从网络上装，适用于网速较快的情况，“Install from Local Directory”是从本地安装，适用于下载了“本地安装版”版本的用户，“Download Without Installing”是不直接安装，只从服务器上下载文件。

如果大家下载的是“网络安装版”版本，这里可以选择“Install from Internet”；如果大家下载的是“本地安装版”版本，这里可以选择“Install from Local Directory”；如果大家仅需要下载文件（方便以后再次安装），这里可以选择“Download Without Installing”。

这里我们建议下载“本地安装版”版本，使用“Install from Local Directory”。

(4) 单击“下一步”按钮，我们需要设置Cygwin的安装路径，必须使用默认的安装目录，也就是“C:\cygwin”，如果使用非默认的安装目录，很有可能会导致有些包安装不成功，给后期编写代码或编译程序带来麻烦。

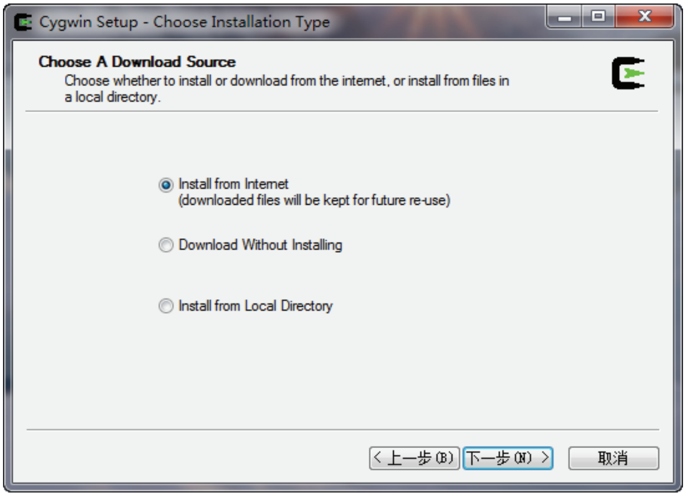


图4-6 Cygwin的安装方式选择页

(5) 单击“下一步”按钮，安装程序会给出Packages的选择界面，此时需要根据自己的需要安装对应的Packages，如果硬盘空间够大的话，建议选择All（全部）。笔者第一次安装的时候就是没有看清这一步，结果没有把GCC装进去，导致没法编译文件。双击这个树形控件的选择界面的某个节点，就可以改变它的状态，有Default（默认）、Install（安装）、Uninstall（卸载）、Reinstall（重新安装）四种状态。默认情况下是Default状态，不过很多工具的Default状态都是不安装。笔者在这里将All这一行上后面的Default改成Install，全部安装，以免后患，防止以后交叉编译的时候缺少一些Package。注意，这里的树形控件和Windows下面其他的树形控件不一样，在All上点击和在All这一行后面的Default上点击会有不同的响应。选择All以后的界面如图4-7所示。

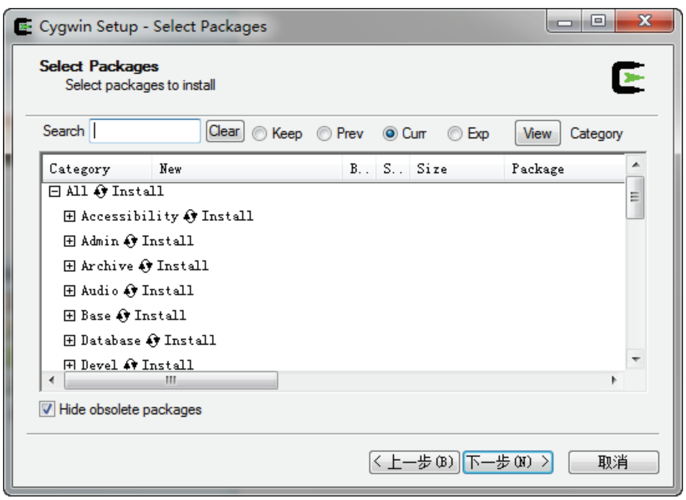


图4-7 Cygwin的选择界面

(6) 单击“下一步”按钮，安装程序会根据我们的设置进行安装。出现图4-8表示Cygwin已经安装成功。

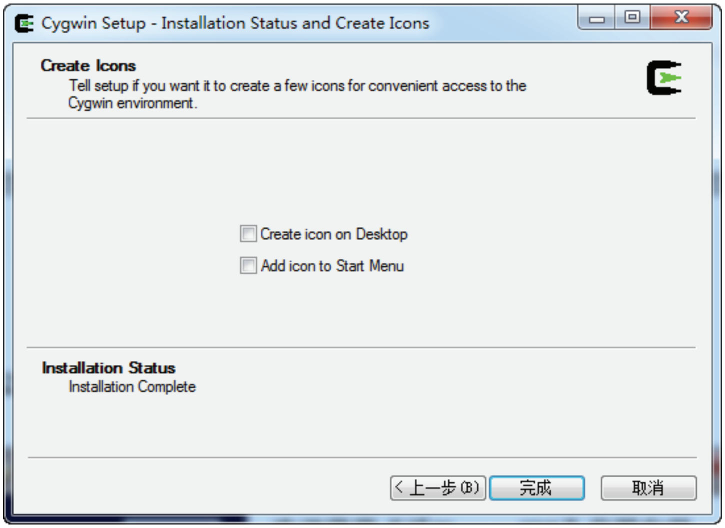


图4-8　Cygwin的完成页

Cygwin是一个在Windows平台上运行的Linux模拟环境，所以Linux命令在上面都可以运行。表4-1给出了几个简单的Linux命令。

表4-1　Linux命令

名　称	描　述
pwd	显示当前路径
cd	改变当前路径，无参数时进入对应用户的home目录
ls	列出当前目录下的文件
ps	列出当前系统进程
kill	杀死某个进程
mkdir	建立目录
rmdir	删除目录
rm	删除文件
mv	文件改名或目录改名
man	联机帮助

由于Linux下命令大多都有很多参数，可以组合使用，所以，每当你不会或者记不清楚该用哪个参数、哪个开关的时候，可以用man来查找。比如，我想查找ls怎么使用，可以键入

```
$ man ls
```

4.2.3 安装iOS环境

iOS的环境包括了iOS的开发环境Xcode和iOS的SDK，目前iOS的环境只能在苹果公司的Mac操作系统上使用，因此需要准备一台Mac的机器，但是每个人都准备两台电脑（一台Windows操作系统的，一台Mac操作系统的）是不怎么现实的。这就体现了跨平台开发引擎的好处：在Windows操作系统上开发，在Mac操作系统上交叉编译发布，完全不需要为每位开发者配置Mac电脑。笔者这里Mac系统选择的是10.6.4版本，Xcode是3.2.5版本，iOS的SDK是4.2版本。下面我们就开始安装iOS环境。

(1) 首先下载Xcode3.2.5和iOS4.2的SDK，从苹果公司的网站上下载安装文件xcode_3.2.5_and_ios_sdk_4.2_final.dmg。双击该安装文件，安装程序会进入安装界面，如图4-9所示。其中“About Xcode and iOS SDK”是一个PDF格式的帮助文件，“Xcode and iOS SDK”则是真正的安装文件。

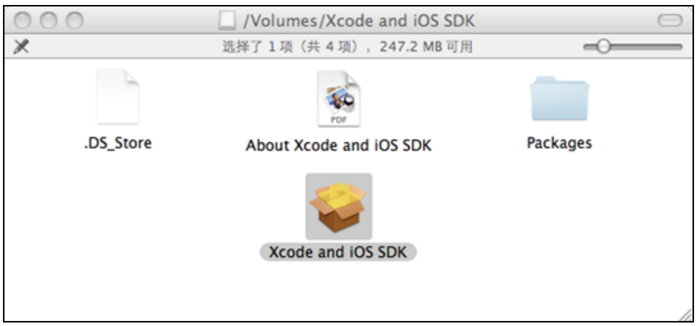


图4-9 iOS的安装界面

(2) 双击安装文件“Xcode and iOS SDK”，出现一个介绍的界面，如图4-10所示。

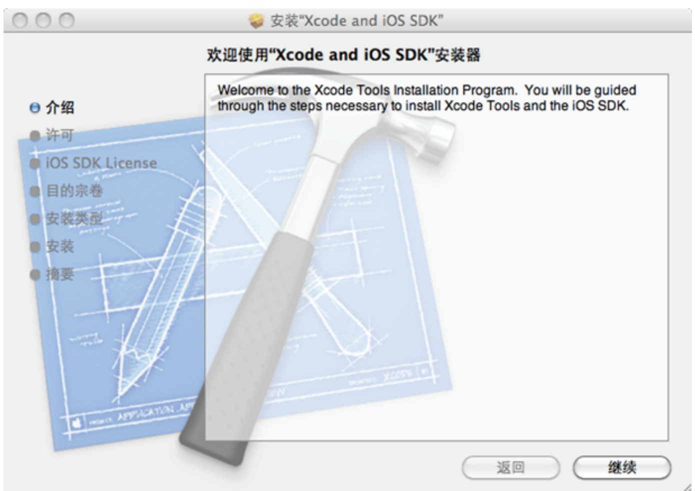


图4-10 iOS的介绍界面

(3) 单击“继续”按钮，安装程序开始进入许可协议的处理，许可协议包括中文协议和英文协议两份（如果不同意该许可协议，则不允许安装程序）。分别同意两份许可协议。中文许可协议的界面如图4-11所示，英文许可协议的界面如图4-12所示。



图4-11　iOS的中文许可协议界面



图4-12　iOS的英文许可协议界面

(4) 两份许可协议都同意之后进入组件选择的界面，如图4-13所示。其中“Xcode Toolset”必须安装，其余可以不安装。“iOS SDK”就是iOS的SDK，“Documentation”则是一些开发需要的文档。

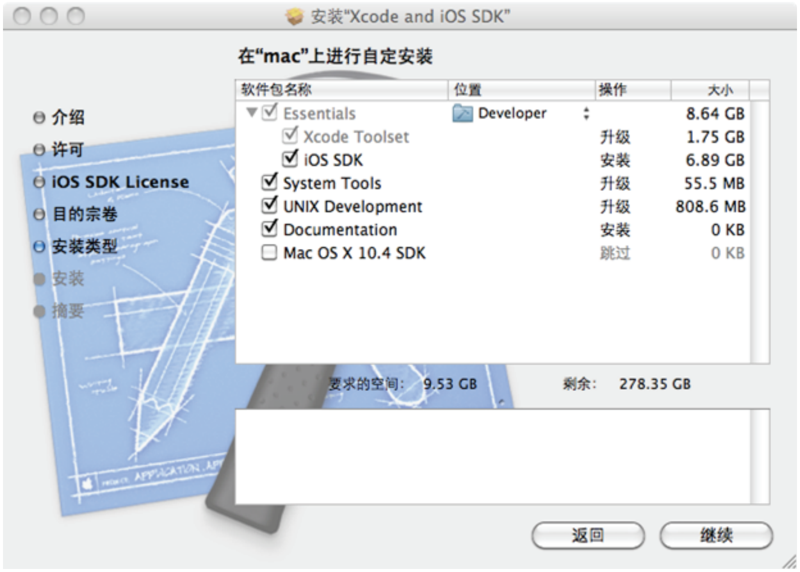


图4-13　iOS的组件选择界面

(5) 单击“继续”按钮，安装程序进入下一步，主要是设置安装目录，如图4-14所示。为了方便起见，一般都使用默认设置。当然也可以通过“更改安装位置”按钮来修改安装目录。

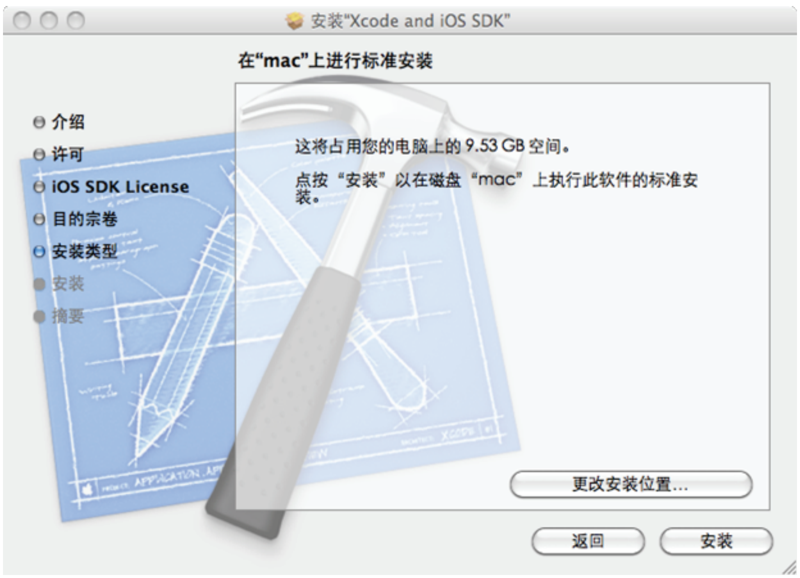


图4-14　iOS的安装目录设置界面

(6) 单击“安装”按钮，安装程序会出现授权的界面，如图4-15所示。



图4-15 iOS的授权界面

(7) 在图4-15所示的界面中输入正确的用户名和密码，安装程序就开始正式安装了。出现图4-16表示Xcode和iOS SDK安装成功。



图4-16 iOS的完成页

4.2.4 安装Android环境

Android的环境包括了Android的SDK（Software Development Kit）和Android的NDK（Native Development Kit），现在分别介绍一下这两个工具。

Android的SDK是Android系统对外提供的基于Java接口的软件工具包。Android的SDK包含了下列组件：Android的模拟器、Android的开发文档、Android示例源码以及相关的调试工具。

Android的NDK是Android系统对外提供的基于C++接口的软件工具包。Android的NDK不能单独存在，需要依赖Android的SDK。它包括下面三个部分。

(1) 提供了一系列的工具，帮助开发者快速开发C或C++的动态库，并能自动将so文件（基于Linux内核的系统的可执行文件）和Java应用一起打包成Android的可执行文件（apk文件）。

(2) 集成了交叉编译器，开发人员只需要简单修改编译脚本文件（指出“哪些文件需要编译”和“编译特性要求”等）就可以创建出so文件。

(3) 提供了一份稳定、功能有限的API头文件声明。

下面我们就开始安装Android环境，首先安装Android的SDK。

(1) 首先从网站上下载Android的SDK的安装文件，笔者写本书的时候，最新的版本为android-sdk_r11-windows.zip。

(2) 解压缩android-sdk_r11-windows.zip得到安装文件SDK Setup.exe，双击该安装文件，安装程序会进入安装界面，如图4-17所示。如果出现错误提示（Failed to fetch URL），那么需要在Setting（设置）内选择“Force https://... sources to be fetched using http://...”选项，选择之后，安装包会重新去下载包的列表，如图4-18所示。

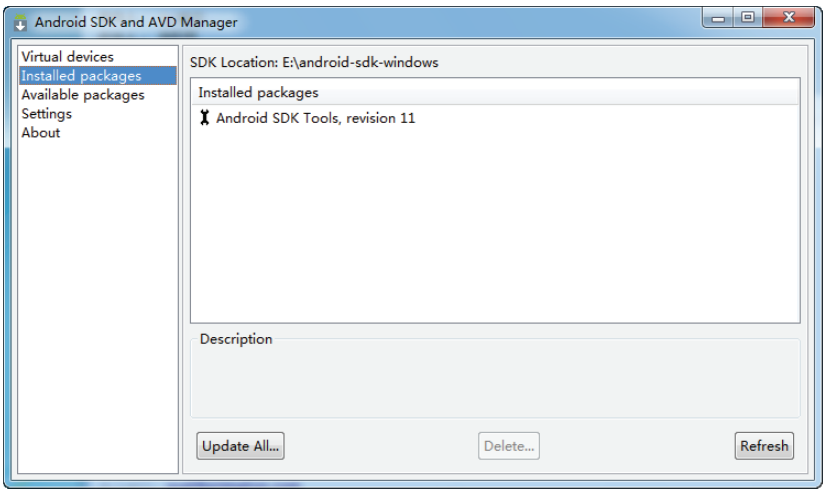


图4-17　Android SDK的安装界面

(3) 此时，应该可以得到图4-19所示的界面。在这里，我们就可以选择需要安装的包。如果网速够快的话，直接选择“Accept All”，然后单击Install按钮。安装程序就开始自动安装选择的包了。这里需要注意的是，安装的过程中没有选择路径的地方，安装程序默认将下载的SDK包放在当前目录中。

(4) 接下来的时间就是等待安装程序安装完成了。

Android的SDK安装完成以后，就可以安装Android的NDK了。

(1) 首先从网站上下载Android的NDK的安装文件，笔者写本书的时候，最新的版本为android-ndk-r5c-windows.zip。

(2) 将下载好的android-ndk-r5c-windows.zip文件解压缩到任何目录即可，作者选择的是C:\cygwin\android-ndk-r5c这个目录。

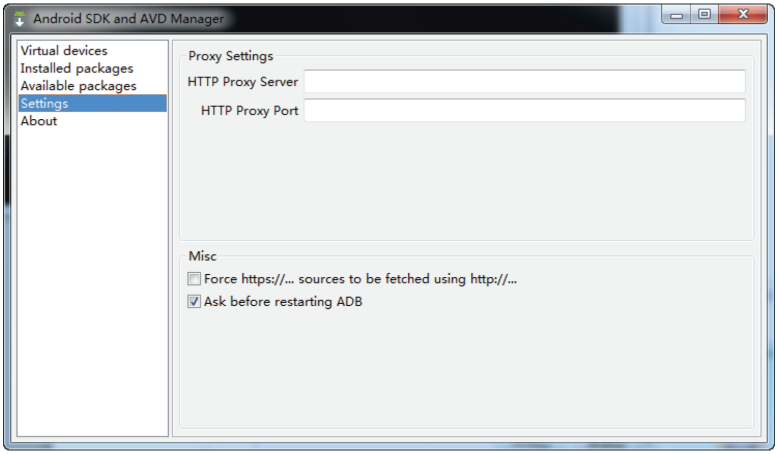


图4-18　Android SDK的出现错误提示时的安装界面

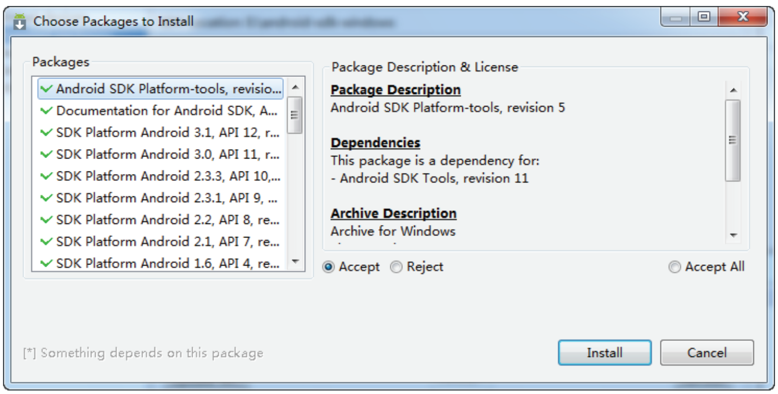


图4-19　Android SDK的包选择页

4.2.5　安装沃Phone环境

沃Phone的环境包括了沃Phone的SDK、TOPS Builder和沃Phone助手，下面我们先分别介绍一下这几个工具包。

沃Phone的SDK包括一套完整的手机软件开发工具，能够帮助广大开发者在沃Phone平台上开发丰富多彩的手机应用软件。它不但用于开发基于C/C++的手机应用软件，而且还能移植Java手机应用软件，生成Widget手机小插件。沃Phone的 SDK包括以下几部分：

(1) 基于C/C++的沃Phone应用程序开发框架编程接口，包含丰富的GUI控件以及系统功能模

块（TCOM）接口；

(2) Java运行环境和编程接口；

(3) Widget运行环境和编程接口；

(4) 非常逼真的Windows平台上的沃Phone仿真器，用于开发和调试软件；

(5) 一系列完善的开发辅助工具，包括沃Phone工程向导、TOPS Builder、沃Phone助手、TMK3、TzdStrConv_V1等；

(6) 各类开发文档，包括沃Phone 应用开发指南和沃Phone SDK 接口说明；

(7) 实用的安装打包工具，开发者在完成沃Phone 应用软件开发之后，通过安装工具打包，即可在沃Phone平台上安装运行。

本书选择的沃Phone的SDK版本为WOPhone-SDK_2011-04-27.zip。

TOPS Builder是一款为沃Phone应用程序开发量身定做的资源IDE制作软件，内置简单易用的所见即所得资源IDE编辑器。它使开发者从繁琐的界面代码中解脱出来，只使用一个软件就可以轻松地完成沃Phone应用程序软件的界面制作。TOPS Builder采用.TRG和.TR3格式来保存所制作的界面，使资源文件通用性增强。

沃Phone助手是一个Windows应用软件，它可以帮助开发者使用沃Phone仿真器的一些特性，并向沃Phone系统注册应用。开发者通过沃Phone助手可以快速启用沃Phone OS仿真器。

下面我们就开始安装沃Phone环境，首先安装沃Phone的SDK。

(1) 首先从网站上下载WOPhone-SDK_2011-04-27.zip。

(2) 解压缩WOPhone-SDK_2011-04-27.zip得到安装文件WOPhoneSDK-setup.exe，双击该安装文件，安装程序会进入安装界面，如图4-20所示。



图4-20　沃Phone SDK的安装界面

(3) 单击“下一步”按钮，安装程序会给出“开始菜单”文件夹的设置，如图4-21所示。



图4-21　沃Phone SDK的“开始菜单”设置页

(4) 单击“安装”按钮，安装程序开始安装。出现图4-22表示沃Phone的SDK已经安装成功。



图4-22　沃Phone SDK的完成页

为了统一使用沃Phone的SDK，目前沃Phone的SDK必须放在D:\Work7 目录中，这也是为什么安装程序不让我们自定义安装目录的原因。

沃Phone的SDK安装完成以后，可以安装TOPS Builder和沃Phone助手。

TOPS Builder的安装文件TOPS-Builder-Setup.exe可以在D:\Work7\目录找到。下面开始介绍TOPS Builder的安装。

(1) 双击安装文件TOPS-Builder-Setup.exe，安装程序会进入安装界面，如图4-23所示。

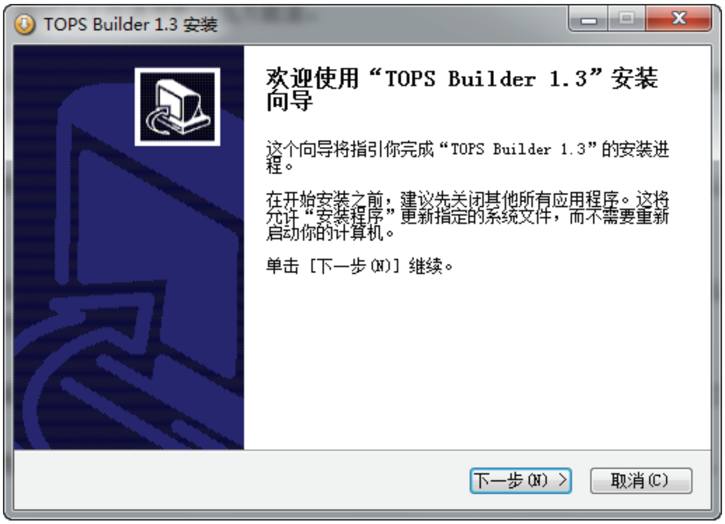


图4-23　TOPS Builder的安装界面

(2) 单击“下一步”按钮，安装程序会给出安装路径的设置，如图4-24所示。

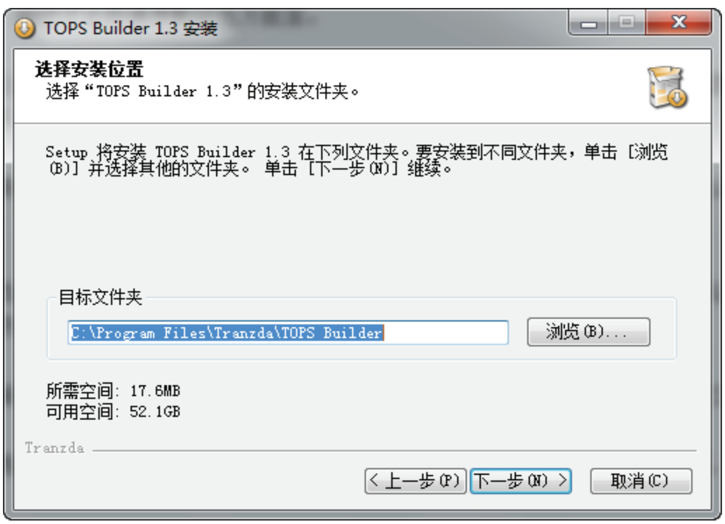


图4-24　TOPS Builder安装的安装路径设置页

(3) 单击“下一步”按钮，安装程序会给出“开始菜单”文件夹的设置，如图4-25所示。

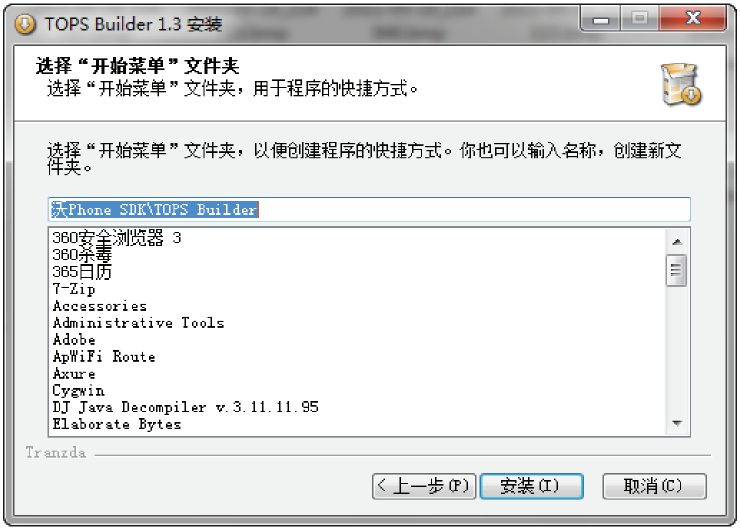


图4-25　TOPS Builder安装的“开始菜单”设置页

(4) 单击“安装”按钮，安装程序开始安装。出现图4-26表示TOPS Builder已经安装成功。



图4-26　TOPS Builder安装的完成页

沃Phone助手的安装文件UTops-setup.exe可以在D:\Work7\目录找到。下面开始介绍沃Phone助手的安装。

(1) 双击安装文件UTops-setup.exe，安装程序会进入安装界面，如图4-27所示。

(2) 单击“下一步”按钮，安装程序会给出安装路径的设置，如图4-28所示。

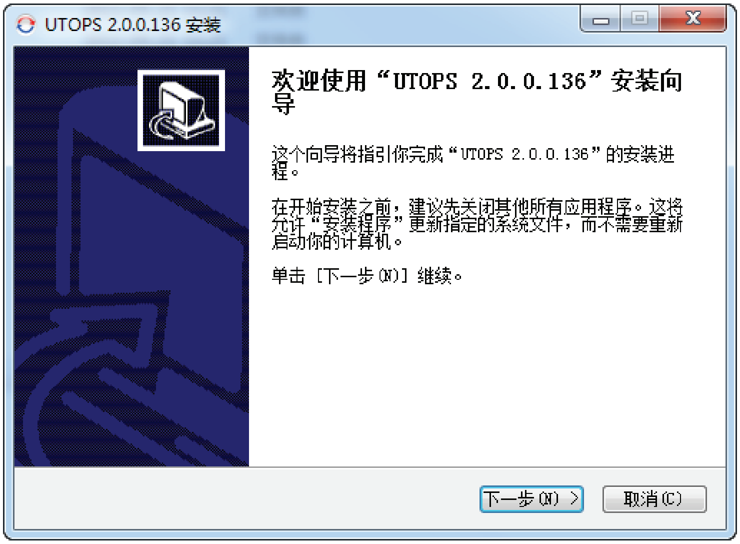


图4-27　沃Phone助手的安装界面

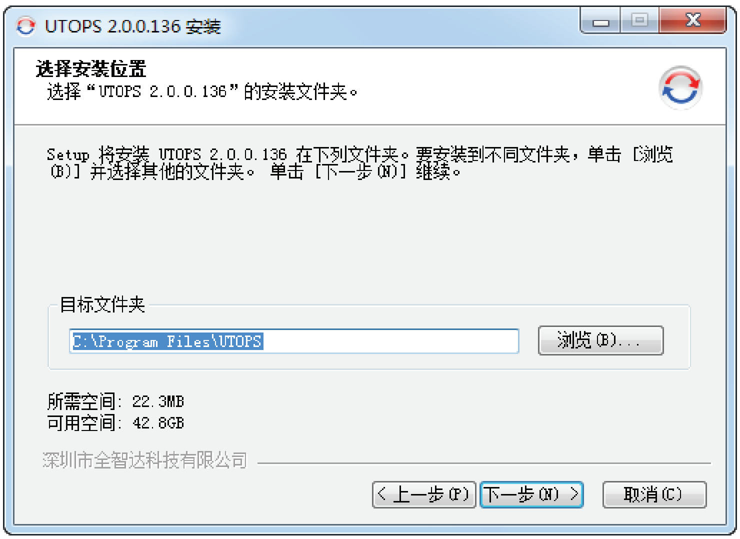


图4-28　沃Phone助手的安装路径设置页

(3) 单击“下一步”按钮，安装程序会给出“开始菜单”文件夹的设置，如图4-29所示。

(4) 单击“安装”按钮，安装程序开始安装。安装过程中需要向系统内注册一些沃Phone的驱动，在Windows 7中会出现图4-30的问题，此时请选择“始终安装此驱动程序软件”。

(5) 出现图4-31表示沃Phone助手已经安装成功。

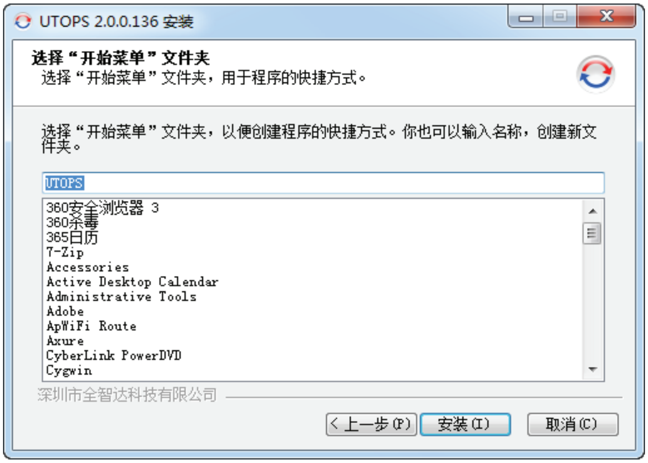


图4-29　沃Phone助手的“开始菜单”设置页



图4-30　沃Phone助手的驱动注册

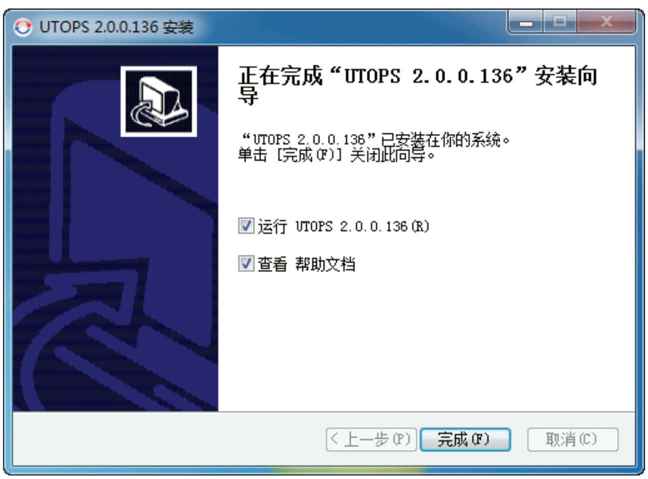


图4-31　沃Phone助手的完成页

沃Phone OS仿真器是沃Phone SDK包含的应用程序手机仿真平台，它在PC上模拟沃Phone手机运行环境，使开发者可以方便地进行手机模拟开发，并能配合Visual Studio 2008的调试器工具进行代码调试。图4-32就是该仿真器的界面。



图4-32　沃Phone OS仿真器

4.2.6　安装cocos2d-x引擎

在第3章里我们已经初步了解了cocos2d-x，现在我们就开始安装cocos2d-x这个游戏引擎。

(1) 首先从网站上下载cocos2d-x。由于这个引擎是开源的，所以你下载到的就是源代码。

(2) 将下载好的cocos2d-x源代码放到D:\Work7\。目录结构如“D:\Work7\cocos2d-x” 即可。

至此，大家已经安装好了跨平台的开发环境，可以稍微喝茶庆祝一下了。

4.3　环境测试之 Hello World 案例

现在我们已经准备好了开发环境，有了这个环境，就可以开始搭建自己的第一个跨平台的游戏。

"Hello World"程序指的是在计算机屏幕上输出"Hello World"这行字符串的计算机程序。一般来说，这是每一种计算机编程语言中最基本、最简单的程序，通常是初学者学习一个新的平台、一种新的语言所编写的第一个程序。同时还可以用来确定该平台、该语言的编译器、开发环境以及运行环境是否已经安装妥当。将输出字符串"Hello World"作为第一个示范程序，现在已经成为编程语言学习的传统。因此，我们也是将"Hello World"作为我们的第一个游戏。

cocos2d-x引擎默认也提供了一个跨平台的"Hello World"实现。下面我们就看看这个"Hello World"游戏在Windows、iOS、Android以及沃Phone上的运行效果。

4.3.1　Windows运行

cocos2d-x引擎针对Windows平台提供了两个解决方案文件，分别是面向vc2008的解决方案文件cocos2d-win32.vc2008.sln和面向vc2010的解决方案文件cocos2d-win32.vc2010.sln。我们这里使用的就是面向vc2008的解决方案文件。

(1) 使用Visual Studio 2008打开解决方案文件cocos2d-win32.vc2008.sln，如图4-33所示。

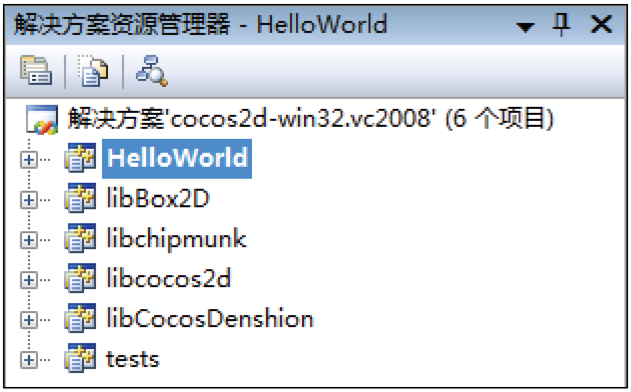


图4-33　解决方案cocos2d-win32.vc2008的6个子项目

(2) 解决方案cocos2d-win32.vc2008一共包含了6个子项目，具体介绍见表4-2。

表4-2　解决方案cocos2d-win32.vc2008中6个子项目的介绍

项目名称	项目描述
libcocos2d	引擎的核心项目
libCocosDenshion	声音解决方案的项目
libBox2D	物理引擎Box2D解决方案的项目
libchipmunk	物理引擎chipmunk解决方案的项目
Hello World	最简单的游戏项目
tests	引擎API的示例项目

(3) 将Hello World项目设置为启动项目。设置的方法是：选中Hello World项目，点击右键后，在弹出菜单上选择“设为启动项目”即可。

(4) 生成Hello World项目。生成的方法是：选中Hello World项目，点击右键后，在弹出菜单上选择“生成”即可。

(5) 运行Hello World项目。运行的方法是：选中Hello World项目，点击右键后，在弹出菜单上选择“调试→启动新实例”即可。运行的效果如图4-34所示。



图4-34　Hello World游戏在Windows上运行

4.3.2　iOS运行

cocos2d-x引擎针对iOS平台没有提供专门的解决方案文件，需要项目的代码和资源文件复制到Mac操作系统上，然后在Mac操作系统上使用开发工具Xcode编译生成iOS的可执行文件，将其部署到iPhone或者iPad上，最终运行的效果如图4-35所示。

基于cocos2d-x引擎的游戏如何在Mac上编译的相关知识将在下一章介绍。



图4-35　Hello World游戏在iOS上运行

4.3.3　Andriod运行

cocos2d-x引擎针对Android平台同样没有提供专门的解决方案文件，需要使用交叉编译工具Cygwin手动交叉编译生成Android的可执行文件（apk文件），然后将apk文件部署到Android手机上，最终运行的效果如图4-36所示。



图4-36　Hello World游戏在Android上运行

关于交叉编译的相关知识将在下一章介绍。

4.3.4　沃Phone运行

cocos2d-x引擎针对沃Phone平台提供了单独的解决方案文件cocos2d-wophone.sln，现在就来介绍如何使用这个解决方案文件运行沃Phone的应用。

(1) 使用Visual Studio 2008打开解决方案文件cocos2d-wophone.sln，如图4-37所示。

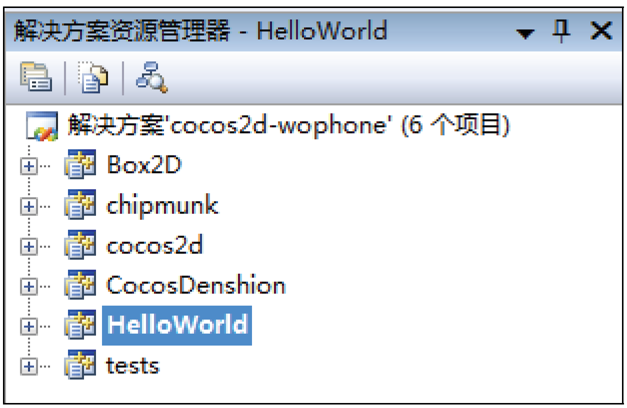


图4-37　解决方案cocos2d-wophone的6个子项目

(2) 解决方案cocos2d-wophone一共包含了6个子项目，具体介绍见表4-3。

表4-3　解决方案cocos2d-wophone中6个子项目的介绍

项目名称	项目描述
cocos2d	引擎的核心项目
CocosDenshion	声音解决方案的项目
Box2D	物理引擎Box2D解决方案的项目
chipmunk	物理引擎chipmunk解决方案的项目
Hello World	最简单的游戏项目
tests	引擎API的示例项目

(3) 将Hello World项目设置为启动项目。设置的方法是：选中Hello World项目，点击右键后，在弹出菜单上选择“设为启动项目”即可。

(4) 生成Hello World项目。生成的方法是：选中Hello World项目，点击右键后，在弹出菜单上选择“生成”即可。

(5) 启动TOPS助手并运行沃Phone OS仿真器。启动的方法是：在开始菜单“沃Phone SDK\TOPS Builder\工具”里，打开“TOPS助手”程序即可。沃Phone助手启动完成后，此时在Windows任务栏通知区域可看到沃Phone助手图标，在图标上点击右键，选择“自动模式启动”，如图4-38所示。



图4-38　自动模式启动

(6) 运行Hello World项目。运行的方法是：等第5步仿真器运行完成后，选中Hello World项目，点击右键后，在弹出菜单上选择“调试→启动新实例”即可。运行的效果如图4-39所示。



图4-39　Hello World游戏在沃Phone上运行

读者按照本步骤运行cocos2d-x引擎在沃Phone上的Hello World时，可能会出现图4-40所示的错误。这时不要惊慌，此错误产生的原因是：沃Phone仿真器集成了旧版本的cocos2d-x，与编译产生的cocos2d-x冲突。解决方案是：将仿真器中自带的cocos2d-x库删除，cocos2d-x包括两个文件（即libcocos2d.dll和libCocosDenshion.dll），cocos2d-x文件目录在D:\Work7\PRJ_TG3\Common\SoftSupport。

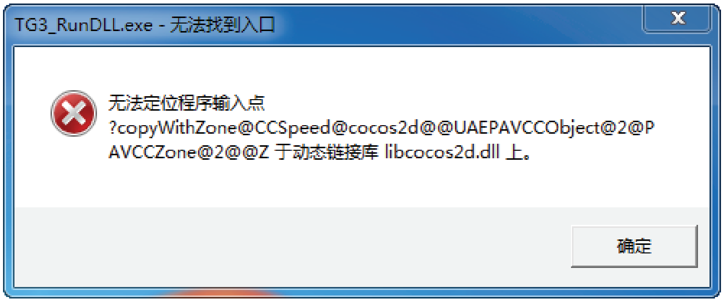


图4-40　沃Phone的仿真器集成旧版本的cocos2d-x时，游戏运行错误

4.4　总结

本章首先介绍如何搭建跨平台的开发环境，包括开发工具Visual Studio、交叉编译Cygwin、iOS环境、Android环境、沃Phone环境和跨平台的游戏引擎cocos2d-x，然后我们将经典的Hello World在这几个平台上运行成功了。

下一章将开始学习cocos2d-x引擎的基础知识以及如何交叉编译游戏等。

第5章

cocos2d-x引擎基础使用

在第4章，我们已经搭建了跨平台的手机游戏开发环境，本章开始学习cocos2d-x引擎。在本章里我们将围绕着游戏讲解开发过程中的各个部分，包括整体架构、图形、动画和布局等。

5.1　整体架构

有这样一个游戏设计：该游戏一共有5个关卡，每个关卡有自己的地图，关卡之间的切换需要有一些转场的动画。在任意一个关卡上都有不同的人物角色（包括英雄和怪物），各人物角色的动作定义如下：英雄可以追打怪物，而怪物被英雄追打时会逃跑。除了英雄和怪物以外，还有一些路人，其中路人甲是围观的、路人乙是打酱油和在一旁做俯卧撑的。除了这些人物之外，天空中还有蓝天和白云。

针对上面这样一个游戏设计，我们会先设计一下游戏的界面流转，请见图5-1。

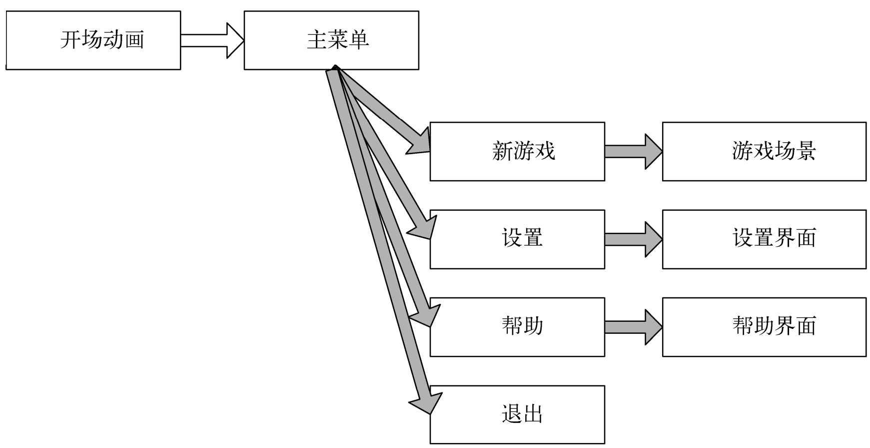


图5-1　游戏的界面流转

现在，我们仔细分析一下图5-1，开场动画主要是显示游戏刚开始时的动画。一般来说显示公司或者工作室的logo，也可以简单描述一下游戏的世界观，甚至还可以简单介绍一下游戏的操作方式。

走完开场动画之后，就进入游戏的主菜单，在这里我们设置了4个功能菜单，分别是新游戏、设置、帮助和退出。这几个功能菜单可以将玩家引导到对应的功能上，下面分别介绍一下。

(1) 新游戏。玩家点击之后进入游戏主场景内，这里就是玩家开始游戏的地方。

(2) 设置。玩家点击之后开始设置一些游戏参数，包括音量的大小、是否震动、是否有一些特殊的效果等。

(3) 帮助。玩家点击之后可以看到一个完整的帮助界面，在这里玩家可以了解到游戏的玩法和技巧等。

(4) 退出。顾名思义，玩家点击之后就退出游戏了。

从这里可以看出玩家玩游戏的过程其实就是在程序设置的场景之间进行跳转，根据一个画面的操作结果跳转到下一个画面。而场景之间的跳转则可以带上很多特效。

那么，我们再看看游戏内的场景会是什么样的一种设计呢？一般来说，我们的设计如图5-2所示。

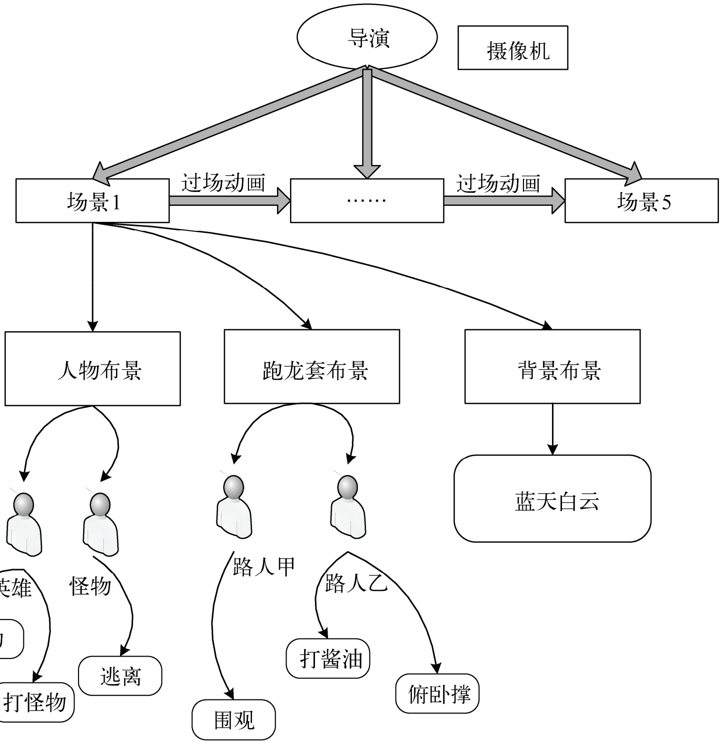


图5-2　游戏主场景设计

我们已经对这个游戏策划做了很好的分析，现在开始将这个游戏策划转变成cocos2d-x的实现框架。

在游戏开发过程中，需要了解几个概念，分别是导演、场景、布景和人物角色。这几个概念和拍电影非常相似，那么我们就拿电影和游戏做类比，一一讲解这几个概念。

(1) 导演（在cocos2d-x引擎中与之对应的类是`CCDirector`）。在电影里，导演是电影艺术创作的组织者和领导者，是把电影文学剧本搬上银幕的总负责人，是用演员表达自己思想的人。导演通过手中的摄像机进行电影的拍摄。那么，在游戏里，导演就是游戏创作的组织者和领导者，是把游戏策划变成游戏的总指挥，导演制定规则让游戏内的场景、布景和人物角色有序地运行。

(2) 场景（在cocos2d-x引擎中与之对应的类是`CCScene`）。在电影里，场景就是电影中的各种场面，各种场面主要由人物活动和背景等构成。那么，在游戏里，场景就是一个个的关卡，关卡主要由布景和人物角色组成。

(3) 布景（在cocos2d-x引擎中与之对应的类是`CCLayer`）。在电影里，布景就是每个场面里面的背景，例如有些场面里面需要的布景是一个办公室，而另外一些场面里面需要的布景则是一座山。那么，在游戏里，布景就是每个关卡里面的背景，同样，不同的关卡需要的背景也是完全不一样的。

(4) 人物角色（在cocos2d-x引擎中与之对应的类是`CCSprite`）。在电影里，人物角色就是电影的人物，既包括电影里的正面人物，也包括电影里的反面人物，既包括主角，也包括跑龙套的。那么，在游戏里，人物角色就包括了游戏的主玩家和其他玩家了。其他玩家既可以是另一个真实的玩家，也可以是游戏虚拟出来的玩家。

(5) 动作（在cocos2d-x引擎中与之对应的类是`CCAction`）。在电影里，动作都是演员的运动，例如走路和射击等。那么，在游戏里，动画就是游戏内人物角色的动作了。

现在，把这几个概念汇总到表5-1中。

表5-1　游戏概念与cocos2d-x引擎中类的对应关系

概　　念	cocos2d-x引擎中对应的类
导演	`CCDirector`
摄像机	`CCCamera`
场景	`CCScene`
布景	`CCLayer`
人物角色	`CCSprite`
动作	`CCAction`

在介绍了游戏概念与cocos2d-x引擎中类的对应关系之后，我们开始把图5-2的游戏主场景设计变成cocos2d-x实现的设计图，请看图5-3。

这张图也就是cocos2d-x实现一个游戏的基本架构了。当然，这个架构不是针对cocos2d-x引擎全新发明的，它同样适合其他平台上的游戏。

任何一款游戏都是通过这些概念组建起来的，而游戏的复杂程度也就决定了这些对象之间关系的复杂程度。接下来我们开始具体地讲解这几个概念以及它们之间的关系。

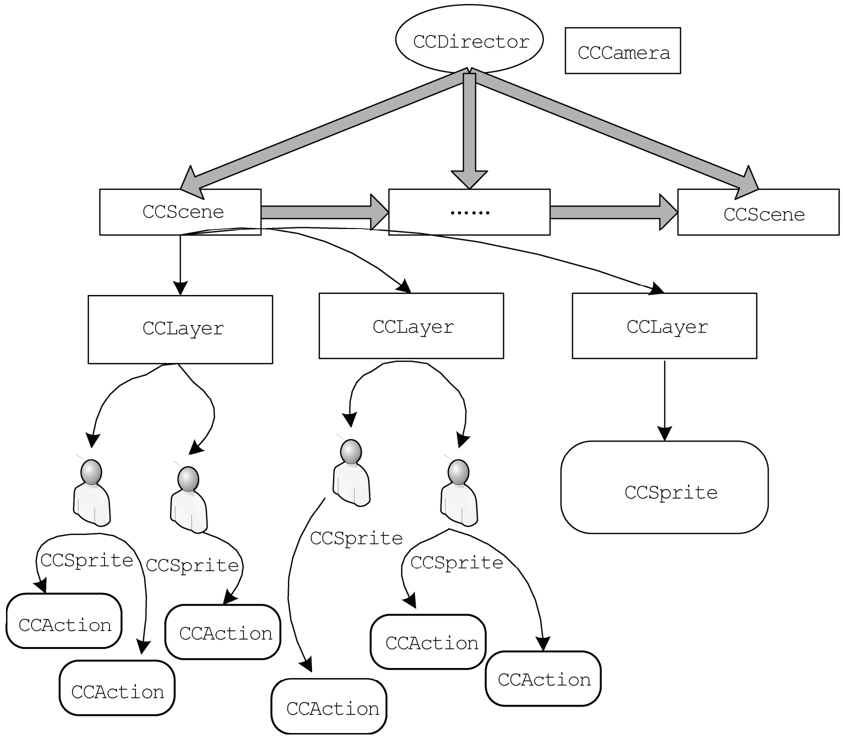


图5-3　游戏主场景设计之cocos2d-x实现

5.1.1　导演

导演在cocos2d-x引擎里面对应的类为CCDirector。在整个游戏里面一般只有一个导演。游戏开始和结束时都需要调用CCDirector的方法完成游戏初始化或者销毁的工作。CCDirector提供了一些管理场景的方法，比如runWithScene、drawScene、pushScene和replaceScene等，通过这些API可以实现在各个场景之间切换，CCDirector也可以设置或者获取一些系统信息，比如调整一些OpenGL 相关的设置，得到屏幕大小等。

cocos2d-x引擎里面除了提供了CCDirector，还提供了一个CCDisplayLink Director。CCDisplayLinkDirector是一个可以自动刷新的CCDirector。在引擎中，CCDirector和CCDisplayLinkDirector的类结构如图5-4所示。

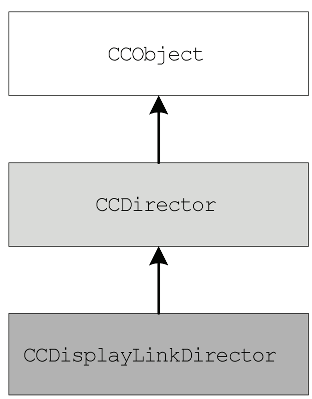


图5-4　CCDirector和CCDisplayLinkDirector的类结构图

CCDirector类的主要函数解释如下：

```
CCScene *getRunningScene(void);//获取当前运行的场景

double getAnimationInterval(void);//获取FPS
void setAnimationInterval(double dValue);//设置FPS

bool isDisplayFPS(void);//是否在屏幕底部显示FPS
void setDisplayFPS(bool bDisplayFPS);//设置在屏幕底部显示FPS
C_GLVIEW *getOpenGLView(void);//获取CCEGLView
void setOpenGLView(CC_GLVIEW *pobOpenGLView);//设置CCEGLView

bool isPaused(void);//是否暂停

ccDirectorProjection getProjection(void);//获取OpenGL的Projection
void setProjection(ccDirectorProjection kProjection);//设置OpenGL的Projection

bool isSendCleanupToScene(void);
//场景替换时是否接收到Cleanup事件。若新的场景是被push进来的，旧的场景不会收到Cleanup事件；若新的
//场景是被替换进来的，旧的场景能够收到Cleanup事件。

CCSize getWinSize(void);//获取OpenGL View的大小，单位为点
CCSize getWinSizeInPixels(void);//获取OpenGL View的大小，单位为像素
CCSize getDisplaySizeInPixels(void);//获取OpenGL View的显示区域大小，单位为像素

void reshapeProjection(CCSize newWindowSize);//改变Projection的大小

CCPoint convertToGL(CCPoint obPoint);//将UIKit坐标体系转化到OpenGL坐标体系
CCPoint convertToUI(CCPoint obPoint);//将OpenGL坐标体系转化到UIKit坐标体系

void runWithScene(CCScene *pScene);//运行场景
void pushScene(CCScene *pScene);//push场景
void popScene(void);//pop场景
void replaceScene(CCScene *pScene);//替换场景
void drawScene(void);//渲染场景

void pause(void);//暂停游戏
void resume(void);//恢复游戏

void stopAnimation(void);//停止动画
void startAnimation(void);//播放动画

void purgeCachedData(void);//删除缓存的数据

void setGLDefaultValues(void);//设置OpenGL的默认值
void setAlphaBlending(bool bOn);//设置是否启用OpenGL的alpha通道
void setDepthTest(bool bOn);//设置是否测试景深

bool enableRetinaDisplay(bool enabled);//设置启用RETINA支持
bool isRetinaDisplay();//是否启用RETINA支持

bool setDirectorType(ccDirectorType obDirectorType);
//设置Director类型，目前cocos2d-x支持4种Director，分别是kCCDirectorTypeNSTimer、
//kCCDirectorTypeMainLoop、kCCDirectorType- ThreadMainLoop、kCCDirectorTypeDisplayLink
```

```
void setPixelFormat(tPixelFormat kPixelFormat);//设置CCEGLView的像素格式
tPixelFormat getPiexFormat(void);//获取CCEGLView的像素格式

CCDirector *sharedDirector(void);//返回CCDirector实例
```

在本章一开始，我们介绍了一个游戏的整体架构：不同的场景由不同的层组成，每个层又包括自己的人物角色或者布景。用户玩游戏的过程其实就是在操作每个层上的人物角色或者功能菜单，这样游戏就可以在不同的场景中切换。

说到这里，我们应该知道导演（CCDirector）对象的作用了：人物角色或者背景不应该依赖层、层不应该依赖场景、场景不应该依赖整个游戏。导演对象是流程的总指挥，它负责游戏全过程的场景切换。这也是典型的面向对象和分层的设计原则。

在cocos2d-x里面，在游戏的任何时间，只有一个场景对象实例处于运行状态。该对象可以作为当前游戏内容对象的整体包容对象。

5.1.2 摄像机

摄像机在cocos2d-x引擎里面对应的类为CCCamera。CCCamera在cocos2d-x中比较重要，每一个节点（CCNode）都需要使用CCCamera。当节点发生缩放、旋转和位置变化等时，都需要覆盖CCCamera，让这个节点通过CCCamera重新渲染。

在引擎中，CCCamera的类结构如图5-5所示。

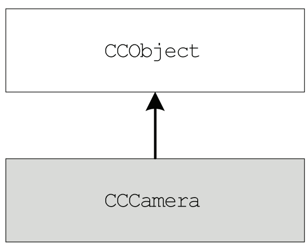


图5-5　CCCamera在cocos2d-x引擎中的类结构图

CCCamera类的主要函数解释如下：

```
void init(void);//初始化函数
void setEyeXYZ(float fEyeX, float fEyeY, float fEyeZ);//设置Eye的坐标
void setCenterXYZ(float fCenterX, float fCenterY, float fCenterZ);//设置Center的坐标
void setUpXYZ(float fUpX, float fUpY, float fUpZ);//设置Up的坐标
void getEyeXYZ(float *pEyeX, float *pEyeY, float *pEyeZ);//获取Eye的坐标
void getCenterXYZ(float *pCenterX, float *pCenterY, float *pCenterZ);
//获取Center的坐标
void getUpXYZ(float *pUpX, float *pUpY, float *pUpZ);//获取Up的坐标
```

有了摄像机（CCCamera），节点才会被渲染成大家可以看到的，例如背景和人物角色等。

5.1.3 场景

场景在cocos2d-x引擎里面对应的类为CCScene。在cocos2d-x引擎中，CCScene中存放的是需要渲染的布景、人物角色和菜单，它可以作为一个整体，一起渲染，一起销毁，一起被场景切换使用。

在cocos2d-x引擎中，CCScene的类结构如图5-6所示。

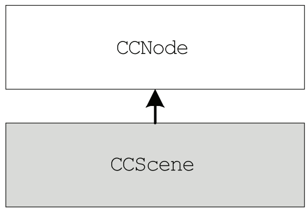


图5-6　CCScene在cocos2d-x引擎中的类结构图

CCScene类的主要函数解释如下：

```
bool init();//初始化函数
static CCScene *node(void);//生成CCScene
```

看了上面的代码，估计大家都很奇怪，为什么CCScene只有初始化和生成CCScene的代码而没有其他方法呢？其实主要还是因为目前在引擎中CCScene承担的是一个容器的功能。游戏开发时把多个需要渲染的对象放在CCScene里面统一管理，包括创建、销毁和场景切换等。那么用什么方法才能把对象添加到CCScene里面呢？这时我们需要分析一下CCNode。

在cocos2d-x引擎中，CCNode的类结构如图5-7所示。

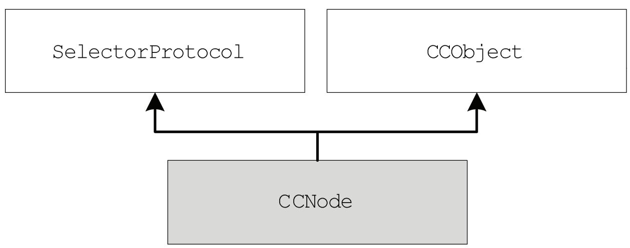


图5-7　CCNode在cocos2d-x引擎中的类结构图

CCNode类的主要函数解释如下：

```
static CCNode *node(void);//生成CCNode

virtual void onEnter();//进入节点时的回调函数
virtual void onEnterTransitionDidFinish();//进入节点后的回调函数
virtual void onExit();//离开节点时的回调函数

virtual void addChild(CCNode *child);//增加节点
```

```
virtual void addChild(CCNode *child, int zOrder);//增加节点
virtual void addChild(CCNode *child, int zOrder, int tag);//增加节点

void removeFromParentAndCleanup(bool cleanup);//删除父节点中的当前节点并清除动作及回调函数等
virtual void removeChild(CCNode *child, bool cleanup);//删除节点
void removeChildByTag(int tag, bool cleanup);//删除节点
virtual void removeAllChildrenWithCleanup(bool cleanup);//删除节点并清除动作及回调函数等

CCNode *getChildByTag(int tag);//根据tag名称获取CCNode
virtual void reorderChild(CCNode *child, int zOrder);//根据zOrder重新排序

CCAction *runAction(CCAction *action);//运行动作
void stopAllActions(void);//停止所有的动作
void stopAction(CCAction *action);//根据action停止动作
void stopActionByTag(int tag);//根据tag名称停止动作
CCAction *getActionByTag(int tag);//根据tag名称获取动作
int numberOfRunningActions(void);//正在运行的动作的总数

void schedule(SEL_SCHEDULE selector);//定义一个定时器
void schedule(SEL_SCHEDULE selector, ccTime interval);//定义一个定时器
void unschedule(SEL_SCHEDULE selector);//取消一个定时器
void unscheduleAllSelectors(void);//取消所有的定时器
void resumeSchedulerAndActions(void);//恢复定时器和动作
```

因为CCScene继承CCNode，所以CCScene拥有了CCNode的一些方法，例如addChild和removeChild方法，这样就可以通过addChild方法把渲染的对象添加到CCScene里面了。

5.1.4　布景

布景在cocos2d-x引擎里面对应的类为CCLayer（层）。

每个游戏场景中都可以有很多层，每一层负责各自的任务，例如专门负责显示背景、专门负责显示道具和专门负责显示人物角色等。在每一层上面可以放置不同的元素，包括文本、精灵和菜单等。

通过层以及层与层之间的组合关系，我们就能够让游戏显示出各种各样的界面了。当然为了能够看到每一层上的东西，很多层都设置为透明或者半透明的，否则大家只能看到最上面一层了。

在引擎中，CCLayer的类结构如图5-8所示。

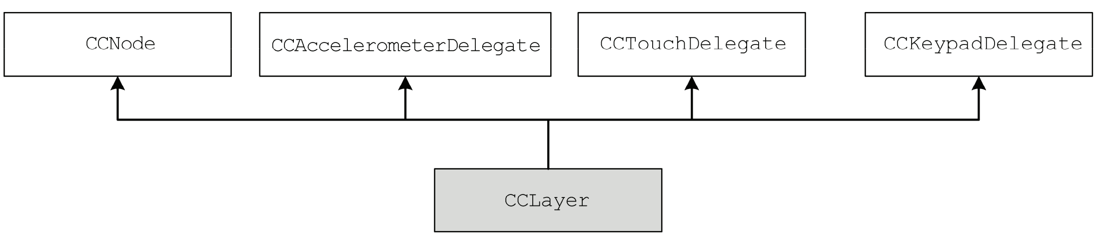


图5-8　CCLayer在cocos2d-x引擎中的类结构图

CCLayer类的主要函数解释如下：

```
bool init();//初始化函数
static CCLayer *node(void);//生成CCLayer

virtual void onEnter();//进入时的回调函数
virtual void onExit();//离开时的回调函数
virtual void onEnterTransitionDidFinish();//进入后的回调函数

virtual bool ccTouchBegan(CCTouch *pTouch, CCEvent *pEvent);//触屏事件
virtual void ccTouchesBegan(CCSet *pTouches, CCEvent *pEvent);//触屏事件
virtual void ccTouchesMoved(CCSet *pTouches, CCEvent *pEvent);//触屏事件
virtual void ccTouchesEnded(CCSet *pTouches, CCEvent *pEvent);//触屏事件
virtual void ccTouchesCancelled(CCSet *pTouches, CCEvent *pEvent);//触屏事件

virtual void destroy(void);//销毁
virtual void keep(void);//保持

virtual void KeypadDestroy();//关闭键盘的回调函数
virtual void KeypadKeep();//打开键盘的回调函数
```

有了层（CCLayer），游戏就可以显示出各种各样的界面了。

5.1.5 人物角色

人物角色在cocos2d-x引擎里面对应的类为CCSprite（精灵）。精灵是整个游戏开发处理的主要对象，天上的飞机、地上的坦克、玩家控制的人物等都是精灵。甚至随机飘过的一片云、飞过的一只鸟也是精灵。从技术上讲，精灵就是一个可以不断变化的图片。这些变化包括位置变化、旋转、放大、缩小和运动等。

在引擎中，CCSprite的类结构如图5-9所示。

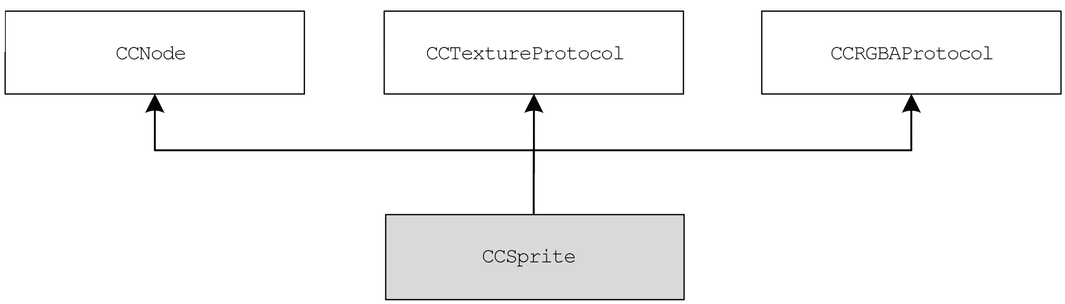


图5-9　CCSprite在cocos2d-x引擎中的类结构图

CCSprite类的主要函数解释如下：

```
bool init(void);//初始化函数

virtual void draw(void);//绘制
```

```
static CCSprite *spriteWithTexture(CCTexture2D *pTexture);//根据纹理生成CCSprite
static CCSprite *spriteWithTexture(CCTexture2D *pTexture, CCRect rect);
//根据纹理生成CCSprite
static CCSprite *spriteWithTexture(CCTexture2D *pTexture, CCRect rect, CCPoint offset);
//根据纹理生成CCSprite
static CCSprite *spriteWithSpriteFrame(CCSpriteFrame *pSpriteFrame);
//根据SpriteFrame生成CCSprite
static CCSprite *spriteWithSpriteFrameName(const char *pszSpriteFrameName);
//根据SpriteFrame名称生成CCSprite
static CCSprite *spriteWithFile(const char *pszFileName);//根据plist文件生成CCSprite
static CCSprite *spriteWithFile(const char *pszFileName, CCRect rect);
//根据plist文件生成CCSprite
static CCSprite *spriteWithBatchNode(CCSpriteBatchNode *batchNode, CCRect rect);
//根据批量节点生成CCSprite
static CCSprite *spriteWithSpriteSheet(CCSpriteSheetInternalOnly *pSpriteSheet, CCRect rect);
//根据SpriteSheet生成CCSprite

virtual void removeChild(CCNode *pChild, bool bCleanup);//移除子节点
virtual void removeAllChildrenWithCleanup(bool bCleanup);//移除所有的子节点
virtual void reorderChild(CCNode *pChild, int zOrder);//移除子节点
virtual void addChild(CCNode *pChild);//增加子节点
virtual void addChild(CCNode *pChild, int zOrder);//增加子节点
virtual void addChild(CCNode *pChild, int zOrder, int tag);//增加子节点

virtual void setIsRelativeAnchorPoint(bool bRelative);//设置是否相对于参考点
virtual void setPosition(CCPoint pos);//设置位置
virtual void setPositionInPixels(CCPoint pos);//设置位置，单位：像素
virtual void setRotation(float fRotation);//设置旋转
virtual void setScaleX(float fScaleX);//设置缩放
virtual void setScaleY(float fScaleY);//设置缩放
virtual void setScale(float fScale);//设置缩放
virtual void setVertexZ(float fVertexZ);//设置最高点
virtual void setAnchorPoint(CCPoint anchor);//设置锚点
virtual void setIsRelativeAnchorPoint(bool bRelative);//设置是否相对于参考点
virtual void setIsVisible(bool bVisible);//设置是否可见

virtual void setTexture(CCTexture2D *texture);//设置纹理
virtual CCTexture2D *getTexture(void);//获取纹理

bool initWithTexture(CCTexture2D *pTexture);//根据纹理初始化CCSprite
bool initWithTexture(CCTexture2D *pTexture, CCRect rect);//根据纹理初始化CCSprite
bool initWithSpriteFrame(CCSpriteFrame *pSpriteFrame);//根据SpriteFrame初始化CCSprite
bool initWithSpriteFrameName(const char *pszSpriteFrameName);
//根据SpriteFrame名称初始化CCSprite
bool initWithFile(const char *pszFilename);//根据plist文件初始化CCSprite
bool initWithFile(const char *pszFilename, CCRect rect);//根据plist文件初始化CCSprite
bool initWithBatchNode(CCSpriteBatchNode *batchNode, CCRect rect);
//根据批量节点初始化CCSprite
bool initWithSpriteSheet(CCSpriteSheetInternalOnly *pSpriteSheet, CCRect rect);
//根据SpriteSheet初始化CCSprite
```

有了精灵（CCSprite），游戏就不再是静止的，而是充满生机的，有各种人物、花草和道具了。

5.1.6 动作

动作在cocos2d-x引擎里面对应的类为CCAction。CCAction主要用于人物角色等发生动作时使用，例如紧身肉搏、远程射击和贴近对话等。

在引擎中，CCAction的类结构如图5-10所示。

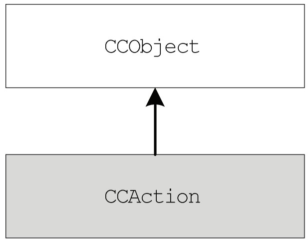


图5-10 CCAction在cocos2d-x引擎中的类结构图

CCAction类的主要函数解释如下：

```
virtual CCObject *copyWithZone(CCZone *pZone);//克隆对象

virtual bool isDone(void);//动作是否已经完成

virtual void startWithTarget(CCNode *pTarget);//设置动作关联的对象，动作运行前调用该方法
virtual void stop(void);//停止动作
virtual void step(ccTime dt);//动作的间隔时间
virtual void update(ccTime time);//动作的执行内容

CCNode *getTarget(void);//获取动作关联的对象
void setTarget(CCNode *pTarget);//设置动作关联的对象

inline CCNode *getOriginalTarget(void);//获取动作的目标对象
inline void setOriginalTarget(CCNode *pOriginalTarget);//设置动作的目标对象

inline int getTag(void);//获取tag
inline void setTag(int nTag)//设置tag

static CCAction *action();//生成CCAction
```

有了动作（CCAction），人物角色在游戏内才能发生动作，例如紧身肉搏、远程射击和贴近对话等。

5.2 目录结构

前面介绍了使用cocos2d-x引擎开发游戏的整体架构。现在我们来看一下cocos2d-x引擎的目录结构，如图5-11所示。

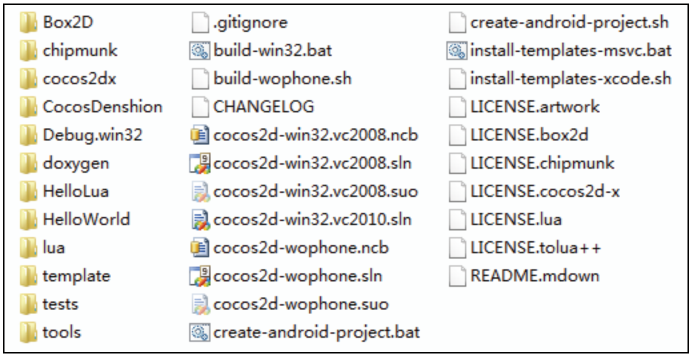


图5-11　cocos2d-x引擎的目录结构

目录结构的具体介绍如下。

- Box2D：物理引擎Box2D的相关源文件。
- chipmunk：物理引擎chipmunk的相关源文件。
- cocos2dx：cocos2d-x引擎的核心部分，存放了引擎大部分的源文件。
- CocosDenshion：声音模块的相关源文件。
- Debug.win32：在Windows上的调试输出目录。
- doxygen：生成doxygen项目文档时需要的配置文件。
- HelloLua：在游戏中使用Lua的示例代码。
- HelloWorld：测试代码HelloWorld。
- lua：脚本语言Lua的相关源文件。
- template：包括编译iOS和Android等平台游戏时需要的配置文件。
- tests：cocos2d-x引擎所有API的示例代码。
- tools：包括“tolua的配置文件”和“xcode4的模板生成工具”。
- build-win32.bat、build-wophone.sh：编译cocos2d-x引擎的Windows项目和沃Phone项目的脚本。
- cocos2d-win32.vc2008.sln、cocos2d-win32.vc2010.sln：Windows项目的解决方案文件。
- cocos2d-wophone.sln：沃Phone项目的解决方案文件。
- create-android-project.bat、create-android-project.sh：创建空的Android项目的脚本。
- install-templates-msvc.bat、install-templates-xcode.sh：给Visual Studio安装cocos2d-x模板的脚本。
- LICENSE.artwork：使用的第三方组件（artwork）的许可文件。
- LICENSE.box2d：使用的第三方组件（box2d）的许可文件。
- LICENSE.chipmunk：使用的第三方组件（chipmunk）的许可文件。
- LICENSE.cocos2d-x：cocos2d-x引擎的许可文件。
- LICENSE.lua：使用的第三方组件（lua）的许可文件。
- LICENSE.tolua++：使用的第三方组件（tolua++）的许可文件。

5.3 坐标体系

在一般的图形系统中，使用屏幕的左上角作为原点（*x*轴等于0，*y*轴等于0），如图5-12所示。Windows系统使用的就是这种坐标体系，而且很多游戏引擎都是使用这种坐标体系。

不过由于cocos2d-x引擎使用OpenGL ES图形库进行渲染，而OpenGL采用的不是图5-12所示的坐标体系。在OpenGL坐标体系中，原点在左下角，如图5-13所示。因此cocos2d-x引擎采用的是OpenGL坐标体系，如图5-13所示。

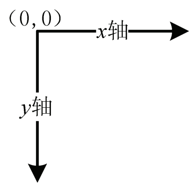


图5-12　坐标体系

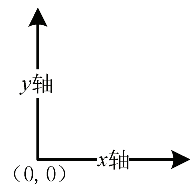


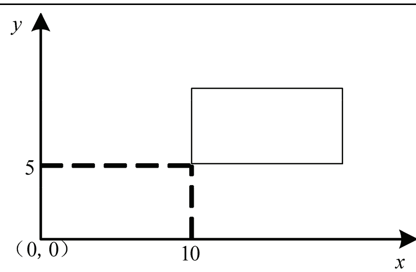
图5-13　OpenGL坐标体系

现在，我们来仔细看一下OpenGL坐标体系。在OpenGL坐标体系中有两个非常重要的参数，即锚点（anchor point）和位置（position）。在cocos2d-x引擎中，大部分需要显示的对象都继承于节点（`CCNode`），我们就以一个矩形为例讲解锚点和位置两个概念。

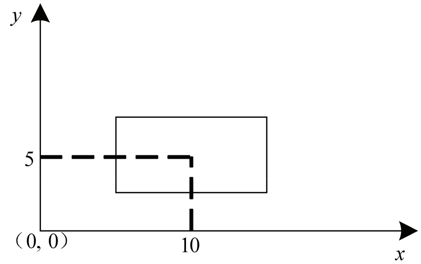
- **锚点**。矩形的位置参考点，这是相对于矩形本身的坐标系来说的。例如锚点为(0.5, 0.5)，而矩形的大小是200 × 100，那么这个锚点相对于矩形的坐标是(100, 50)。缺省的锚点位置为(0.5, 0.5), 也就是在矩形的中间。
- **位置**。顾名思义，就是指矩形被放置的位置。

了解了锚点和位置的概念后，我们看一个具体的例子。表5-2描述了位置相同、锚点不同的两个矩形的示意图。

表5-2　位置相同、锚点不同的两个矩形的示意图

矩形描述	示　意　图
锚点：(0, 0) 位置：(10, 5)	

（续）

矩形描述	示　意　图
锚点：(0.5,0.5) 位置：(10,5)	

5.4　跨平台常量

cocos2d-x引擎是一个跨平台的游戏引擎，那么它到底是如何做到跨平台的呢？

首先cocos2d-x引擎是一个基于C++语言的引擎，而苹果的iOS、Google的Android、中国联通的沃Phone、微软的Windows以及苹果最新推出的AirPlay这些平台都支持直接使用C++进行编写软件或游戏，那么我们就可以做一套公共的接口，然后根据平台选择不同的代码进行编译。这就是cocos2d-x引擎能够跨平台的原因。

cocos2d-x定义了一组常量，这组常量用于表示不同的平台，如表5-3所示。

表5-3　跨平台常量

常量名称	常　量　值	描　　述
CC_PLATFORM_UNKNOWN	0	未知平台，也就是平台不在cocos2d-x引擎的支持列表之内
CC_PLATFORM_IOS	1	苹果的iOS平台，包括iPhone和iPad
CC_PLATFORM_ANDROID	2	谷歌的Android平台
CC_PLATFORM_WOPHONE	3	中国联通的沃Phone平台
CC_PLATFORM_WIN32	4	微软的Windows平台
CC_PLATFORM_AIRPLAY	5	苹果的AirPlay平台

有了上面这组跨平台常量之后，我们就可以针对不同的平台写不同的代码，大致代码如下：

```
//iOS平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_IOS)
    //需要在iOS平台上执行的相关代码……
#endif

//Android平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_ANDROID)
    //需要在Android平台上执行的相关代码……
```

```
#endif

//沃Phone平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_WOPHONE)
    //需要在沃Phone平台上执行的相关代码……
#endif

//Windows平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_WIN32)
    //需要在Windows平台上执行的相关代码……
#endif

//AirPlay平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_AIRPLAY)
    //需要在AirPlay平台上执行的相关代码……
#endif

//其他平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_UNKNOWN)
    //给出不支持的提示……
#endif
```

5.5 图形

游戏中的每一个元素都是图形，包括菜单、图片、标签、人物、花草树木、墙壁和道具等。那么在游戏中是如何将这些元素有序地管理起来并一一渲染的呢？

从5.1节的介绍中，我们知道游戏的界面存在很多层（CCLayer），元素被添加到层里面之后，cocos2d-x引擎在渲染层的时候会同时渲染层里面的元素，这样就可以渲染出游戏界面了。cocos2d-x引擎目前提供以下元素。

(1) **标签（CCLabelTTF）**。在游戏界面中显示一段文字，CCLabelTTF的使用代码如下：

```
//创建CCLabelTTF
CCLabelTTF *label = CCLabelTTF::labelWithString("测试文字", "Arial", 28);

//增加到当前层中
this.addChild(label);
```

(2) **图片（CCSprite）**。在游戏界面中显示一个图片，CCSprite的使用代码如下：

```
//创建图片
CCSprite *leftMenuBg = CCSprite::spriteWithFile("btn_left_bottom_bg.png");

//增加到当前层中
this.addChild(leftMenuBg);
```

(3) **菜单（CCMenu）及菜单项（CCMenuItem）**。在游戏界面中显示一个可以选择的菜单。菜单项主要包括标签式菜单项（CCMenuItemLabel）、图片式样（CCMenuItemImage）、组合式（CCMenuItemToggle），CCMenuItemImage的使用代码如下：

```
//创建开始游戏菜单项
CCMenuItemFont *start = CCMenuItemFont::itemFromString("开始游戏", this, "");

//创建退出游戏菜单项
CCMenuItemFont *exit = CCMenuItemFont::itemFromString("退出游戏", this, "");

//创建菜单
CCMenu *menu = CCMenu::menuWithItems( start, exit, NULL);

//增加到当前层中
this.addChild(menu);
```

(4) **进度条（CCProgressTimer）**。

```
//创建进度条
CCProgressTimer *progressTimer = CCProgressTimer::progressWithFile("grossinis_sister1
.png");

//增加到当前层中
this.addChild(progressTimer);
```

(5) **瓦片地图图片集（CCTileMapAtlas）**。在游戏里显示一个用瓦片地图图片集设计的瓦片地图。CCTileMapAtlas的使用代码如下：

```
//创建瓦片地图图片集
CCTileMapAtlas *map = CCTileMapAtlas::tileMapAtlasWithTileFile("TileMaps/tiles.png",
"TileMaps/levelmap.tga", 16, 16);

//增加到当前层中
this.addChild(map);
```

(6) **TMX瓦片地图（CCTMXTiledMap）**。在游戏里显示一个TMX瓦片地图。CCTMXTiledMap的使用代码如下：

```
//创建TMX瓦片地图，地图文件orthogonal-test3.tmx
CCTMXTiledMap *map = CCTMXTiledMap::tiledMapWithTMXFile("orthogonal-test3.tmx");

//增加到当前层中
this.addChild(map);
```

在游戏中，如果对上面的这些元素进行设计并合理地组合一起，那么游戏将会很美。

5.6　动作

刚才我们介绍了图形，有了图形游戏就有了好看的界面。那么游戏内的元素如何变化呢？例如，一个人物角色如何从游戏的A点运动到B点呢？这需要引入游戏引擎的另一个概念——动作。

在cocos2d-x引擎中，动作（CCAction）定义了在节点上进行的通用操作，它不依赖于节点，但是在运行时需要指定节点为目标。动作最直观的好处就是可以实现很多动画效果。

CCAction类是所有动作的基类，在引擎中CCAction的类结构如图5-14所示。

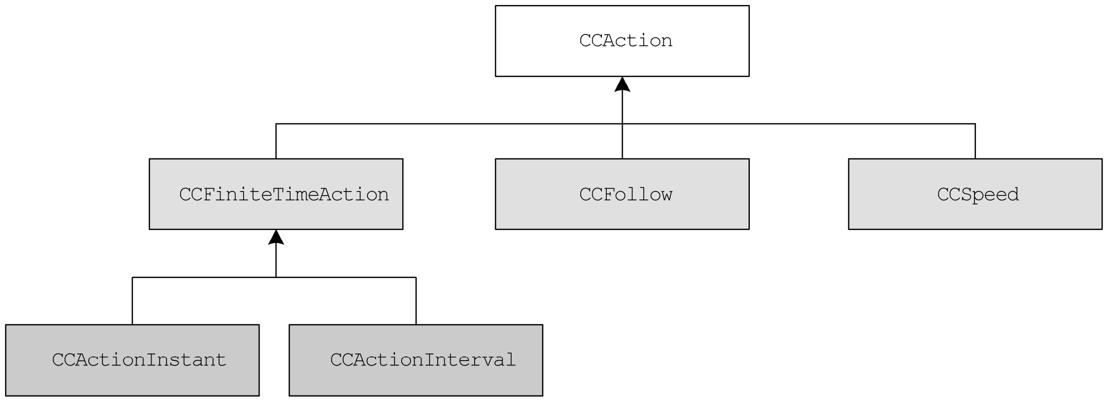


图5-14　CCAction在cocos2d-x引擎中的类结构图

从图5-14中，我们看出动作主要分为两大类：瞬时动作（CCActionInstant）和延时动作（CCActionInterval）。

- **瞬时动作**。主要特点是动作的执行不需要花费时间，马上就能完成动作的执行。这些动作的基类是CCActionInstant。cocos2d-x引擎直接提供的瞬时动作见表5-4。

表5-4　cocos2d-x引擎中提供的瞬时动作

瞬时动作名称	描　　述
CCCallFunc	执行函数
CCFlipX	X翻转
CCFlipY	Y翻转
CCHide	隐藏
CCPlace	设置位置
CCReuseGrid	重用网格
CCShow	显示
CCStopGrid	停止网格
CCToggleVisibility	可见切换

- **延时动作**。主要特点是动作的执行需要花费时间。这些动作的基类是CCActionInterval。cocos2d-x引擎直接提供的延时动作见表5-5。

表5-5　cocos2d-x引擎中提供的延时动作

延时动作名称	描　　述	延时动作名称	描　　述
CCBezierBy	贝塞尔曲线	CCMoveTo	移动到
CCBlink	闪烁	CCMoveBy	移动
CCDelayTime	延迟	CCRotateTo	旋转到
CCFadeTo	变暗到	CCRotateBy	旋转

（续）

延时动作名称	描　　述	延时动作名称	描　　述
CCFadeIn	由无变亮	CCScaleTo	放大到
CCFadeOut	由亮变无	CCScaleBy	放大
CCJumpTo	跳跃到	CCTintTo	色调变化到
CCJumpBy	跳跃	CCTintBy	色调变化
CCSequence	序列	CCSplitRows	拆分行
CCSpawn	同步	CCSplitCols	拆分列
CCReverse	动作逆向	CCRepeat	有限次数重复
CCReverseTime	时间逆向	CCRepeatForever	无限次数重复
CCAnimate	动画		

从5.1节的介绍中，我们了解了动作（CCAction）的主要函数解释，而且coocs2d-x引擎给我们提供了非常多的瞬时动作和延时动作。现在我们就看几个实际的使用场景。

场景一：精灵移动到坐标为(100,100)的位置。

```
//创建精灵
CCSprite *sprite = CCSprite::spriteWithFile("action_sprite1.png");

//创建移动到的动作
CCActionInterval *actionTo = CCMoveTo::actionWithDuration(2, CCPointMake(100, 100));

//动作执行
sprite->runAction( actionTo);
```

场景二：精灵放大到1.5倍。

```
//创建精灵
CCSprite *sprite = CCSprite::spriteWithFile("action_sprite2.png");

//创建放大到的动作
CCActionInterval *actionTo = CCScaleTo::actionWithDuration( 2, 1.5f);

//动作执行
sprite->runAction( actionTo);
```

场景三：精灵跳跃到坐标(100, 100)的位置。

```
//创建精灵
CCSprite *sprite = CCSprite::spriteWithFile("action_sprite3.png");

//创建跳跃到的动作
CCActionInterval *actionTo = CCJumpTo::actionWithDuration(2, CCPointMake (100,100),
50, 4);

//动作执行
sprite->runAction( actionTo);
```

5.7 菜单

一般的游戏都需要提供菜单功能。菜单主要是让玩家选择菜单项对应的功能。为了实现菜单，我们可以通过图片、图片的点击事件来实现。除了使用这种自定义的方案外，cocos2d-x引擎还内置了一种菜单的实现方式。菜单的实现类为CCMenu，菜单项的实现类为CCMenuItem。现在我们分别来看看菜单和菜单项。菜单是一组菜单项的集合。菜单项是一个具体的游戏功能。

在引擎中CCMenu的类结构如图5-15所示，CCMenuItem的类结构如图5-16所示。

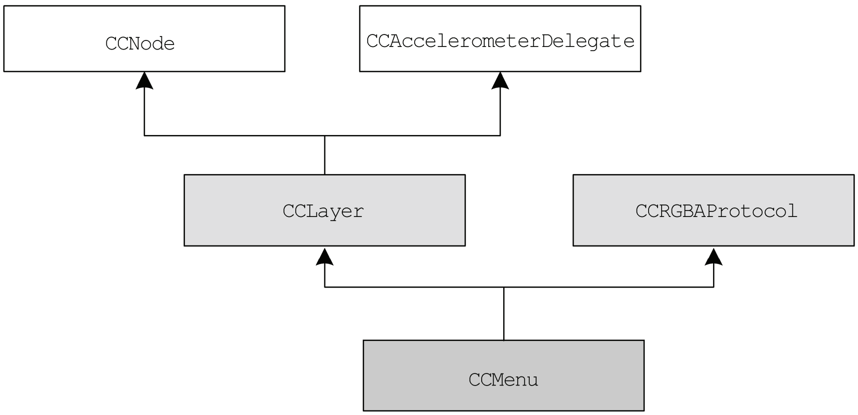


图5-15 CCMenu在cocos2d-x引擎中的类结构图

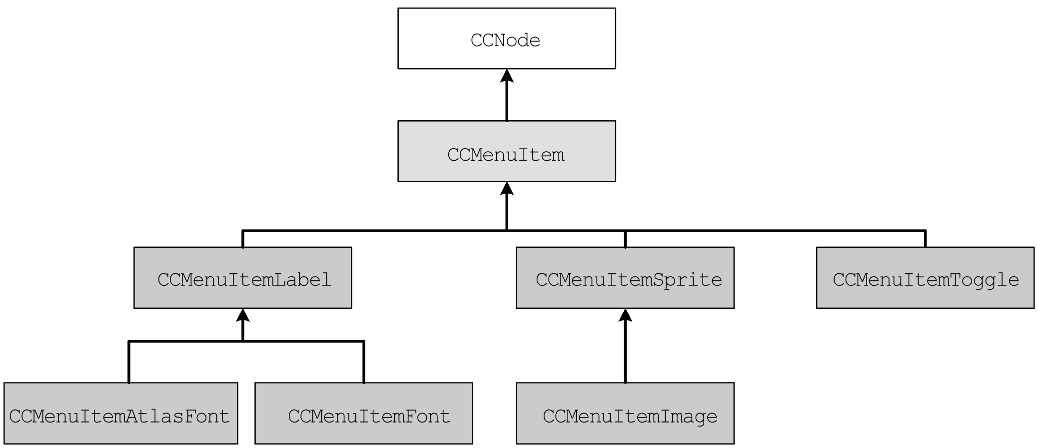


图5-16 CCMenuItem在cocos2d-x引擎中的类结构图

有了上述CCMenu和CCMenuItem的相关介绍，我们就可以在游戏中增加菜单了，下面是几个菜单的示例代码。

```
//设置字体大小
CCMenuItemFont::setFontSize(24);

//创建开始游戏菜单项
CCMenuItemFont *start = CCMenuItemFont::itemFromString("开始游戏", this, "");

//创建退出游戏菜单项
CCMenuItemFont *exit = CCMenuItemFont::itemFromString("退出游戏", this, "");

//创建菜单
CCMenu *menu = CCMenu::menuWithItems( start, exit, NULL);
```

5.8　事件

用户在玩游戏的时候，需要用到手机键盘、虚拟键盘和触摸屏对游戏中人物和背景进行控制。那么，我们就需要在程序中捕获这些事件。cocos2d-x引擎支持3种事件派发，分别是CCKeypadDispatcher、CCIMEDispatcher和CCTouchDispatcher。这3种事件派发分别对应手机键盘、虚拟键盘和触摸屏。

CCKeypadDispatcher对应着手机键盘的事件派发，主要函数解释如下：

```
static CCKeypadDispatcher *sharedDispatcher();//返回CCKeypadDispatcher
static void purgeSharedDispatcher();//释放CCKeypadDispatcher

void addDelegate(CCKeypadDelegate *pDelegate);//增加CCKeypadDelegate代理
void removeDelegate(CCKeypadDelegate *pDelegate);//移除CCKeypadDelegate代理
void forceAddDelegate(CCKeypadDelegate *pDelegate);  //增加CCKeypadDelegate代理
void forceRemoveDelegate(CCKeypadDelegate *pDelegate);//移除CCKeypadDelegate代理

bool dispatchKeypadMSG(ccKeypadMSGType nMsgType);//派发ccKeypadMSGType事件
```

CCIMEDispatcher对应着虚拟键盘的事件派发，主要函数解释如下：

```
static CCIMEDispatcher *sharedDispatcher();//返回CCIMEDispatcher

void dispatchInsertText(const char *pText, int nLen);//派发增加字符事件
void dispatchDeleteBackward();//派发删除字符事件

void dispatchKeyboardWillShow(CCIMEKeyboardNotificationInfo& info);
//虚拟键盘显示过程的事件
void dispatchKeyboardDidShow(CCIMEKeyboardNotificationInfo& info);//虚拟键盘显示后的事件
void dispatchKeyboardWillHide(CCIMEKeyboardNotificationInfo& info);
//虚拟键盘关闭过程的事件
void dispatchKeyboardDidHide(CCIMEKeyboardNotificationInfo& info);//虚拟键盘关闭后的事件

const char * getContentText();//获取内容
```

CCTouchDispatcher对应触摸屏的事件派发，主要函数解释如下：

```
bool isDispatchEvents(void);//是否可以派发事件
void setDispatchEvents(bool bDispatchEvents);//设置是否派发事件

void addStandardDelegate(CCTouchDelegate *pDelegate, int nPriority);
//增加一个CCTouchDelegate代理
void addTargetedDelegate(CCTouchDelegate *pDelegate, int nPriority, bool bSwallowsTouches);
//增加一个CCTouchDelegate代理

void removeDelegate(CCTouchDelegate *pDelegate);//移除一个CCTouchDelegate代理
void removeAllDelegates(void);//移除所有CCTouchDelegate代理

void setPriority(int nPriority, CCTouchDelegate *pDelegate);
//设置CCTouchDelegate代理的优先级

void touches(CCSet *pTouches, CCEvent *pEvent, unsigned int uIndex);

virtual void touchesBegan(CCSet *touches, CCEvent *pEvent);//开始的事件
virtual void touchesMoved(CCSet *touches, CCEvent *pEvent);//移动的事件
virtual void touchesEnded(CCSet *touches, CCEvent *pEvent);//结束的事件
virtual void touchesCancelled(CCSet *touches, CCEvent *pEvent);//取消的事件
```

5.9 变量自动释放

学过C++和Java两种语言的人大部分都认为Java语言比C++要简单，尤其是在变量的内存管理上，C++要复杂得多。我们现在来看看这两种语言在变量的内存释放上的处理机制。

(1) 在Java语言中定义一个变量，不需要开发者关心变量的内存释放，因为Java虚拟机会在变量不再使用的时候自动释放内存空间。

(2) 在C++语言中定义一个变量，当不再使用这个变量的时候，需要手动进行内存释放，否则会产生内存溢出问题。

上述这个内存释放问题，对于一个C++的新人来说是一种折磨。因为你很难明白什么时候需要释放内存，什么时候不需要释放内存。

一般比较成熟的框架都在框架级别解决了这个问题，同样，cocos2d-x引擎也解决了这个问题。如果对cocos2d-x引擎源代码比较清楚的话，就会发现基本上所有的类都是继承CCObject这个类，而这个类就解决了上述问题。

首先在CCObject中定义一个变量m_uRefrence，如果变量m_uRefrence大于1，那么在释放CCObject的时候需要执行释放（delete），释放的同时将m_uRefrence减去1，如果其他代码继续需要释放CCObject，那么由于m_uRefrence等于0（表示已经释放），就不需要释放了。上述解决方案的示例代码如下：

```
//释放CCObject
void CCObject::release(void)
{
```

```
        assert(m_uRefrence > 0);
        --m_uRefrence;

        if (m_uRefrence == 0)
        {
            delete this;
        }
    }
```

5.10　总结

本章主要介绍cocos2d-x引擎的基础使用，包括cocos2d-x引擎的整体架构、目录结构和开发过程中常用的一些常量介绍等，还介绍了cocos2d-x引擎在图形、动作、菜单、事件和字体等方面的知识。

本章主要围绕coocs2d-x引擎的基础进行介绍。在下一章里，我们将主要介绍cocos2d-x引擎的一些高级特性，包括物理引擎、粒子系统和声音模块等。

第6章

cocos2d-x之高级特性

通过上一章的介绍，我们应该已经掌握了cocos2d-x引擎的整体架构、基础知识，现在我们就来学习cocos2d-x引擎的一些高级特性。这些高级特性包括物理引擎、粒子系统和声音模块。

6.1 物理引擎

在游戏中，我们有时候需要让整个游戏模拟出一个真实的物理世界，游戏内所有的物体都需要遵循真实自然的规律。例如，人物跳的高度、子弹的飞行轨迹和车辆的颠簸方式等。

要完成一些简单的物理效果，例如牛顿定律里面的加速度、减速度，可以直接通过编写程序实现。但是想模拟出一个完整的真实物理世界，我们就需要使用专门的物理引擎了。毕竟术业有专攻嘛。

物理引擎通过为物体赋予真实的物理属性的方式来计算运动、旋转和碰撞反映。物理引擎使用对象属性来模拟物体行为，这不仅可以得到更加真实的结果，而且对于开发人员来说也比较容易掌握。好的物理引擎允许有复杂的机械装置，像球形关节、轮子、气缸或者铰链。有些也支持非刚性体的物理属性，比如流体。

cocos2d-x引擎支持两个物理引擎，分别是Box2D和Chipmunk。不过由于Chipmunk的文档和例子相对较少，而Box2D有很完善的文档，再加上作者对Box2D相对更熟悉点，所以作者选择Box2D作为游戏的物理引擎。

首先，我们需要了解物理引擎Box2D的几个常见概念，它们分别是世界（World）、刚体（Body）、刚体定义（BodyDef）、关联（Fixture）、关联定义（FixtureDef）、形状（Shape）、链接（Joint）以及链接定义（JointDef）。

6.1.1 世界

世界代表了一个遵守某些物理定律的空间。世界的定义结构如下（在Box2D/Dynamics/b2World.h里面）：

```
class b2World
{
```

```
public:
    //初始化函数
    b2World(const b2Vec2& gravity, bool doSleep);

    //析构函数，所有的物理对象被销毁，所有的内存被释放
    ~b2World();

    //注册一个碰撞检测之前的过滤器
    void SetContactFilter(b2ContactFilter *filter);

    //注册一个碰撞检测事件的监听器
    void SetContactListener(b2ContactListener *listener);

    //创建一个刚体
    b2Body *CreateBody(const b2BodyDef *def);

    //销毁一个刚体
    void DestroyBody(b2Body *body);

    //创建一个链接
    b2Joint *CreateJoint(const b2JointDef *def);

    //销毁一个链接
    void DestroyJoint(b2Joint *joint);

    //获取刚体列表
    b2Body *GetBodyList();

    //获取链接列表
    b2Joint *GetJointList();

    //获取碰撞检测的列表
    b2Contact *GetContactList();

    //获取刚体总数
    int32 GetBodyCount() const;

    //获取链接总数
    int32 GetJointCount() const;

    //获取碰撞检测的总数
    int32 GetContactCount() const;

    //设置重力向量
    void SetGravity(const b2Vec2& gravity);

    //获取重力向量
    b2Vec2 GetGravity() const;

    //获取世界是否被锁定
    bool IsLocked() const;
};
```

创建世界时需要完成两个步骤，一是生成一个重力向量，二是根据重力向量生成世界对象，示例代码如下：

```
//生成一个重力向量
b2Vec2 gravity;
gravity.Set(0.0f, -10.0f);

//根据重力向量生成世界对象
b2World *world = new b2World(gravity, true);
```

6.1.2 刚体及刚体定义

刚体代表一个质点，因此它只有位置，没有大小。物理引擎Box2D把刚体分为三种类型。

(1) **静态刚体（Static Body）**。静态刚体没有质量，没有速度，如果你想改变它的位置，只能通过代码修改。

(2) **棱柱刚体（Prismatic Body）**。棱柱刚体没有质量，但是可以有速度，引擎会根据速度计算并更新它的位置。

(3) **动态刚体（Dynamic Body）**。动态刚体有质量也可以有速度，这是我们最常用的刚体类型了。

在编写程序时物理引擎Box2D要求先定义一个描述类，然后通过这个描述类创建某个对象。所以创建刚体就需要先定义一个刚体的信息（刚体定义），然后通过世界的相关方法创建刚体。当刚体被创建时刚体定义中的信息会被复制，所以在刚体创建完成后刚体定义还可以重复用，但是要记得不需要刚体定义时要释放掉。

“刚体”和“刚体定义”的定义结构如下（在Box2D/Dynamics/b2Body.h里面），创建刚体时需要完成两个步骤，一是生成一个刚体定义，二是根据刚体定义生成刚体，示例代码如下：

```
//生成一个刚体定义
b2BodyDef groundBodyDef;
groundBodyDef.position.Set(0, 0);

//根据刚体定义生成刚体
b2Body *groundBody = world->CreateBody(&groundBodyDef);
```

6.1.3 形状

形状通过关联附加到刚体上，这样刚体就具有了视觉上的外形，由于刚体上可以有多个关联，所以刚体上也就可能有多个形状，Box2D称为组合形状（Compound Shapes），这样可以构成更复杂的形状。

“形状”的定义结构如下（在Box2D/Collision/Shapes/b2Shape.h里面），创建形状的代码比较简单，示例代码如下：

```
//生成一个形状
b2PolygonShape dynamicBox;
dynamicBox.SetAsBox(0.5f, 0.5f);
```

6.1.4　关联及关联定义

关联代表了一种附加在刚体上的属性，刚体上可以有多个关联。创建关联时需要先定义一个关联的信息（关联定义），然后通过世界的相关方法创建关联。当关联被创建时关联定义中的信息会被复制，所以在关联创建完成后关联定义还可以重复用，但是要记得不需要关联定义时要释放掉。

“关联”和“关联定义”的定义结构如下（在Box2D/Dynamics/b2Fixture.h里面）,创建关联时需要完成两个步骤，一是生成一个关联定义，二是根据关联定义生成关联，示例代码如下：

```
//生成一个动态刚体的关联定义
b2FixtureDef fixtureDef;
fixtureDef.shape = &dynamicBox;
fixtureDef.density = 1.0f;
fixtureDef.friction = 0.3f;

//根据关联定义生成关联
b2Fixture *fixture = body->CreateFixture(&fixtureDef);
```

6.1.5　链接及链接定义

链接用来联系多个刚体，使它们产生相互的影响。链接有很多类型，所以必须根据情况选择你需要的链接类型。创建链接时需要先定义一个链接的信息（链接定义），然后通过世界的相关方法创建链接。当链接被创建时链接定义中的信息会被复制，所以在链接创建完成后链接定义还可以重复用，但是要记得不需要链接定义时要释放掉。

“链接”和“链接定义”的定义结构如下（在Box2D/Dynamics/Joints/b2Joint.h里面），创建链接时需要完成两个步骤，一是生成一个链接定义，二是根据链接定义生成链接，示例代码如下：

```
//生成一个链接定义
b2LineJointDef jointDef;
b2Vec2 axis(2.0f, 1.0f);
axis.Normalize();
jointDef.Initialize(ground, body, b2Vec2(0.0f, 8.5f), axis);
jointDef.motorSpeed = 0.0f;
jointDef.maxMotorForce = 100.0f;
jointDef.enableMotor = true;
jointDef.lowerTranslation = -4.0f;
jointDef.upperTranslation = 4.0f;
jointDef.enableLimit = true;

//根据链接定义生成链接
b2Joint *joint = m_world->CreateJoint(&jointDef);
```

6.1.6　使用案例

有了上述介绍的物理引擎Box2D的知识以及相关示例代码，我们就可以在代码中使用这个物理引擎了。示例代码如下：

```
//获取屏幕大小
CCSize screenSize = CCDirector::sharedDirector()->getWinSize();

//定义重力向量
b2Vec2 gravity;
gravity.Set(0.0f, -10.0f);

//根据重力向量生成世界对象
world = new b2World(gravity, true);

world->SetContinuousPhysics(true);

//生成一个刚体定义
b2BodyDef groundBodyDef;
groundBodyDef.position.Set(0, 0);

//根据刚体定义生成刚体
b2Body *groundBody = world->CreateBody(&groundBodyDef);

//生成一个形状
b2PolygonShape groundBox;

//设置形状属性
groundBox.SetAsEdge(b2Vec2(0,0), b2Vec2(screenSize.width/PTM_RATIO,0));

//根据形状生成关联
groundBody->CreateFixture(&groundBox, 0);

//设置形状属性
groundBox.SetAsEdge(b2Vec2(0,screenSize.height/PTM_RATIO),
b2Vec2(screenSize.width/PTM_RATIO,screenSize.height/PTM_RATIO));

//根据形状生成关联
groundBody->CreateFixture(&groundBox, 0);

//生成一个链接定义
b2LineJointDef jointDef;
b2Vec2 axis(2.0f, 1.0f);
axis.Normalize();
jointDef.Initialize(ground, body, b2Vec2(0.0f, 8.5f), axis);
jointDef.motorSpeed = 0.0f;
jointDef.maxMotorForce = 100.0f;
jointDef.enableMotor = true;
jointDef.lowerTranslation = -4.0f;
jointDef.upperTranslation = 4.0f;
jointDef.enableLimit = true;

//根据链接定义生成链接
b2Joint *joint = m_world->CreateJoint(&jointDef);
```

6.2 粒子系统

在游戏中，经常需要做一些动画效果，例如英雄与怪物之间的战斗效果。我们可以通过动画

实现游戏的动画效果。但是，如果需要显示一些更加真实的效果，例如实现更加随机并且好看的烟雾、雷电、雨雪以及火花等效果，就不能通过动画来实现，因为这些效果自身是动态的（随时间迅速变化的），而这种动态变化的效果是由大量微粒组合而形成。正因为如此，很难用一个确定的数学公式来模拟出这些效果。

为了在游戏中实现这些效果，我们必须引进粒子系统。有了粒子系统，游戏才能显得更加真实而富有生命感。在粒子系统中，至少要包括四大部分：大量的粒子对象、每个粒子遵守的规律、每个粒子的随机性和持续更新的粒子状态。按照预先定义好的规则，不断产生出大量粒子，每个粒子都按照定义好的参数进行整体变化和个性变化，大量的粒子效果叠加就可以形成我们所需要的效果。

粒子系统主要分为如下两种模式。

(1) **重力式粒子系统**。这种粒子系统中存在“重力”，就好像地球的万有引力一样，所有的粒子都会受到重力的约束，当然重力的大小是可以自己定义的。

(2) **放射式粒子系统**。这种粒子系统中不存在重力，因此粒子都好像在太空，不再受到地球万有引力的作用。

粒子系统说明了这些粒子要遵守某种规则，而规则是通过一系列参数定义的。这些参数中，有些既可以用于重力式粒子系统，也可以用于放射式粒子系统，而有些只能用于重力式粒子系统或者放射式粒子系统。表6-1是这些参数的简单介绍及适用范围。

表6-1　粒子系统参数的简单介绍及适用范围

名　　称	描　　述	粒子系统	
		重力式	放射式
重心	粒子系统的重心	√	√
速度	粒子的初速度	√	×
方向	粒子的初速方向	√	√
尺寸	粒子的大小	√	√
生命	粒子存在的时间	√	√
颜色	粒子的颜色	√	√
自转角度	粒子是否要绕着自己的轴心旋转	√	√
公转角度	粒子是否要以重心为轴心旋转	×	√
角加速度	粒子的角加速度，如果角加速度不等于零，那么粒子会围绕重心旋转	√	×
线加速度	粒子的线加速度，如果线加速度不等于零，那么粒子的速度会发生变化	√	×
半径	粒子的当前位置和重心的距离	√	√
分组模式	分组模式规定了一个粒子是否跟着重心移动，不分组表示粒子在出生后就不会跟着重心移动了，分组表示如果重心改变了，粒子也会相对改变位置	√	√

表6-1中介绍的参数都可以附加一个随机参数，这样可以保证产生的粒子具有一定的差异。所以可以把粒子系统看成一个状态机，一旦设置好了初始状态和结束状态，那么这个系统里所有的粒子就会从这个初始状态逐渐变化到结束状态。

在cocos2d-x引擎里面，粒子对象的定义结构如下（在文件include/CCParticleSystem.h里面）：

```
typedef struct sCCParticle {
    CCPoint pos;//位置
    CCPoint startPos; //起始位置

    ccColor4F color;//颜色
    ccColor4F deltaColor;//颜色变化

    float size; //大小
    float deltaSize;//大小变化

    float rotation;//旋转角度
    float deltaRotation;//旋转角度的变化

    ccTime timeToLive;//存在时间

    struct {
        CCPoint dir;//运动方向
        float radialAccel;//径向加速度。逃离发射原点的速度
        float tangentialAccel;//切线加速度。围绕发射原点旋转速度
    } modeA;

    struct {
        float angle;//角度
        float degreesPerSecond;//每秒的角度变化
        float radius;//半径
        float deltaRadius;//半径变化
    } modeB;

}tCCParticle;//粒子对象
```

在cocos2d-x里面粒子系统的定义结构如下（在文件include/CCParticleSystem.h里面）：

```
class CCParticleSystem : public CCNode, public CCTextureProtocol
{
protected:
    struct {
        CCPoint gravity;//重力向量
        float speed;//运动速度
        float speedVar;//运动速度的变化率
        float tangentialAccel;//切向加速度
        float tangentialAccelVar;//切向加速度的变化率
        float radialAccel;//径向加速度
        float radialAccelVar;//径向加速度的变化率
    } modeA;

    struct {
        float startRadius;//起始半径
```

```
        float startRadiusVar;//起始半径的变化率
        float endRadius;//结束半径
        float endRadiusVar;//结束半径的变化率
        float rotatePerSecond;//旋转角度
        float rotatePerSecondVar;//旋转角度的变化率
    } modeB;

    tCCParticle *m_pParticles;//粒子的数组
    float m_fEmitCounter;//发射器每秒发射的粒子数

};//粒子系统
```

通过上述粒子对象和粒子系统的对象描述，我们可以看出modeA主要定义一些宏观的属性，而modeB则定义一些微观的属性。modeB的数据与随机数相乘就可以实现随机的效果。CCRANDOM_MINUS1_1是一个随机函数，随机地生成一个介于−1到1之间的数据。

在cocos2d-x引擎中，一共提供了两类粒子系统，分别是重力式的粒子系统（CCParticle-SystemPoint）和放射式的粒子系统（CCParticleSystemQuad），这两个粒子系统都继承于CCParticleSystem。这三者的类结构关系如图6-1所示。

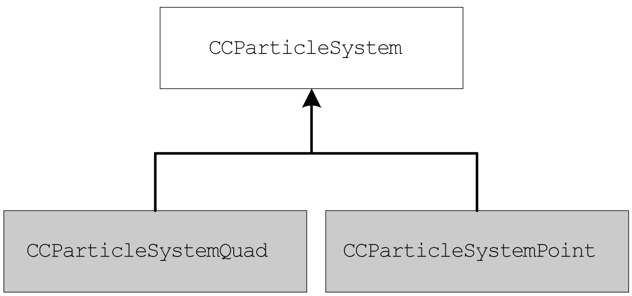


图6-1　CCParticleSystem在cocos2d-x引擎中的类结构关系图

CCParticleSystem主要定义了粒子对象的数据结构以及粒子系统的一些公共方法，CCParticleSystemPoint和CCParticleSystemQuad则根据自身特性增加了一些新的方法。而且CCParticleSystemQuad比CCParticleSystemPoint支持更多的特性：

(1) CCParticleSystemQuad能够围绕轴线进行旋转，而CCParticleSystemPoint忽略了这个属性；

(2) CCParticleSystemQuad支持更多的粒子对象；

(3) CCParticleSystemQuad可以使用scale属性进行缩放。

粒子系统CCParticleSystem的主要函数解释如下：

```
CCParticleSystem();//初始化粒子系统

static CCParticleSystem *particleWithFile(const char *plistFile);
//根据plist文件生成粒子系统
```

```
bool initWithFile(const char *plistFile);//根据plist文件初始化粒子系统
bool initWithDictionary(CCDictionary<std::string, CCObject *> *dictionary);
//根据目录初始化粒子系统
virtual bool initWithTotalParticles(int numberOfParticles);
//根据粒子对象的总数初始化粒子系统

bool addParticle();//增加一个粒子对象
void initParticle(tCCParticle *particle);//初始化一个粒子对象
void stopSystem();//停止粒子系统
void resetSystem();//重置粒子系统
bool isFull();//粒子系统是否已经满了
```

刚才讲述的粒子和粒子系统的对象，相信大家已经理解了。现在我们就开始使用这些对象来实现整个粒子系统的动态效果。现在我们分别看一下CCParticleSystemPoint和CCParticle-SystemQuad这两个粒子系统的示例代码。

6.2.1 重力式粒子系统

重力式粒子系统也就是CCParticleSystemPoint粒子系统，主要特点是这个系统中存在"重力"，就好像万有引力一样，所有的粒子都会受到这个重力的约束，就好像你在空中那必然是会掉向地面一样。重力式粒子系统的使用步骤如下。

第一步，建立一个Point的粒子系统。

```
//建立一个Point的粒子系统
CCParticleSystem *m_emitter = new CCParticleSystemPoint();
```

第二步，产生300个粒子对象。

```
//产生300个粒子对象
m_emitter->initWithTotalParticles(300);
```

第三步，设置粒子对象以及粒子系统的相关参数。

```
//设置粒子的图像，粒子虽然小，但是还是需要有自己的图像，如果没有texture属性，引擎会默认一个图像
m_emitter->setTexture( CCTextureCache::sharedTextureCache()->addImage("test.png") );

//设置粒子系统的持续时间
m_emitter->setDuration(-1);

//设置重力向量
m_emitter->setGravity(CCPointZero);

//设置角度、角度的变化率
m_emitter->setAngle(0);
m_emitter->setAngleVar(360);

//设置径向加速度、径向加速度的变化率
m_emitter->setRadialAccel(70);
m_emitter->setRadialAccelVar(10);

//设置切向加速度、切向加速度的变化率
```

```
m_emitter->setTangentialAccel(80);
m_emitter->setTangentialAccelVar(0);
//设置运动速度、运动速度的变化率
m_emitter->setSpeed(50);
m_emitter->setSpeedVar(10);

//设置粒子位置、粒子位置的变化率
m_emitter->setPosition( CCPointMake( s.width/2, s.height/2) );
m_emitter->setPosVar(CCPointZero);

//设置粒子的存在时间、存在时间的变化率
m_emitter->setLife(2.0f);
m_emitter->setLifeVar(0.3f);

//设置粒子开始时颜色、粒子开始时颜色的变化率
m_emitter->setStartColor({0.5f, 0.5f, 0.5f, 1.0f});
m_emitter->setStartColorVar({0.5f, 0.5f, 0.5f, 1.0f});

//设置粒子结束时颜色、粒子结束时颜色的变化率
m_emitter->setEndColor({0.1f, 0.1f, 0.1f, 0.2f});
m_emitter->setEndColorVar({0.1f, 0.1f, 0.1f, 0.2f});

//设置粒子开始时大小、粒子开始时大小的变化率
m_emitter->setStartSize(1.0f);
m_emitter->setStartSizeVar(1.0f);

//设置粒子结束时大小、粒子结束时大小的变化率
m_emitter->setEndSize(32.0f);
m_emitter->setEndSizeVar(8.0f);

//设置每秒产生的粒子数
m_emitter->setEmissionRate(m_emitter->getTotalParticles()/m_emitter->getLife());
```

第四步，将这个粒子系统设置到背景层上。

```
//将m_emitter设置到m_background这个层上面
m_background->addChild(m_emitter, 1);
```

6.2.2　放射式粒子系统

放射式粒子系统也就是CCParticleSystemQuad粒子系统，主要特点是这个系统中不存在重力，粒子都好像到了太空。那么如果我把重力式粒子系统中的重力设置为0，是否就等于放射式？否，因为这两种模式各自有一些特定参数。放射式粒子系统的使用方式如下。

第一步，建立一个Quad的粒子系统。

```
//建立一个Quad的粒子系统
CCParticleSystem *m_emitter = new CCParticleSystemQuad();
```

第二步，产生300个粒子对象。

```
//产生300个粒子对象
m_emitter->initWithTotalParticles(300);
```

第三步，设置粒子对象以及粒子系统的相关参数。

//设置粒子的图像，粒子虽然小，但是还是需要有自己的图像，如果没有texture属性，引擎会默认一个图像。笔者认为默认的这个图片非常丑

```
m_emitter->setTexture( CCTextureCache::sharedTextureCache()->addImage("test.png") );

//设置粒子系统的持续时间
m_emitter->setDuration(-1);

//设置重力向量
m_emitter->setGravity(CCPointZero);

//设置角度、角度的变化率
m_emitter->setAngle(90);
m_emitter->setAngleVar(360);

//设置运动速度、运动速度的变化率
m_emitter->setSpeed(160);
m_emitter->setSpeedVar(20);

//设置径向加速度、径向加速度的变化率
m_emitter->setRadialAccel(-120);
m_emitter->setRadialAccelVar(0);

//设置切向加速度、切向加速度的变化率
m_emitter->setTangentialAccel(30);
m_emitter->setTangentialAccelVar(0);

//设置粒子位置、粒子位置的变化率
m_emitter->setPosition( CCPointMake(160,240) );
m_emitter->setPosVar(CCPointZero);

//设置粒子的存在时间、存在时间的变化率
m_emitter->setLife(3);
m_emitter->setLifeVar(1);

//设置粒子开始时自旋转速度、粒子开始时自旋转速度的变化率
m_emitter->setStartSpin(0);
m_emitter->setStartSpinVar(0);

//设置粒子结束时自旋转速度、粒子结束时自旋转速度的变化率
m_emitter->setEndSpin(0);
m_emitter->setEndSpinVar(2000);

//设置粒子开始时颜色、粒子开始时颜色的变化率
m_emitter->setStartColor({0.5f, 0.5f, 0.5f, 1.0f});
m_emitter->setStartColorVar({0.5f, 0.5f, 0.5f, 1.0f});

//设置粒子结束时颜色、粒子结束时颜色的变化率
```

```
m_emitter->setEndColor({0.1f, 0.1f, 0.1f, 0.2f});
m_emitter->setEndColorVar({0.1f, 0.1f, 0.1f, 0.2f});

//设置粒子开始时大小、粒子开始时大小的变化率
m_emitter->setStartSize(30.0f);
m_emitter->setStartSizeVar(00.0f);

//设置粒子结束时大小、粒子结束时大小的变化率
m_emitter->setEndSize(30.0f);
m_emitter->setEndSizeVar(00.0f);

//设置每秒产生的粒子数
m_emitter->setEmissionRate(m_emitter->getTotalParticles()/m_emitter->getLife());
```

第四步，将这个粒子系统设置到背景层上。

```
//将m_emitter设置到m_background这个层上面
m_background->addChild(m_emitter, 1);
```

从上面两个示例可以看出粒子系统的使用方法基本相同，唯一不同的就是参数。那么我们如何设计这么多的参数，而且还得让参数设计好之后的粒子系统产生我们想要的效果呢？这就要用到粒子系统的设计工具了。cocos2d-x引擎直接支持的粒子系统设计工具是Particle Designer。

除了上述直接使用参数实现粒子效果外，cocos2d-x引擎还实现几种常用的粒子效果（见表6-2），这几种常用的粒子效果定义在CCParticleExamples.h文件内。

表6-2　cocos2d-x实现的粒子效果

英文名称	类　名
Fire	CCParticleFire
Fireworks	CCParticleFireworks
Sun	CCParticleSun
eGalaxy	CCParticleGalaxy
Flower	CCParticleFlower
Meteor	CCParticleMeteor
Spiral	CCParticleSpiral
Explosion	CCParticleExplosion
Smoke	CCParticleSmoke
Snow	CCParticleSnow
Rain	CCParticleRain

粒子系统能产生大千世界的大部分行为，这些就需要大家一起来研究了，也许还需要很多灵感。

6.3　声音模块

在游戏开发中，声音的地位并不如图形那么重要。我们在开发游戏时会花费大量的精力做图

形的新功能和特效，还有专门的美工设计图形，然而花在声音这一块的精力肯定没有图形多。即使是如此，一个游戏仍然少不了声音的存在。

没有声音的游戏算不上一个好游戏。那么如何让游戏发出声音呢？一般情况下，不同的平台都提供了专门用来播放声音的接口，我们只需要调用这些接口就可以了。在iOS、Android和沃Phone这三个平台上，每个平台都有播放声音的接口，而且这些接口的实现方式都不一样。通过这些平台的接口，游戏就可以播放声音。可是如果这样的话，我们就需要针对不同的平台编写不同的播放声音的代码，这种做法不满足cocos2d-x引擎跨平台的要求。

因此，cocos2d-x引擎提供了一个单独的声音模块，这个模块封装了iOS、Android和沃Phone这三个平台播放声音的接口。这样游戏里面就可以用同一套代码完成不同平台上声音的播放了。声音模块代码的定义结构如下（在文件SimpleAudioEngine.h里面）：

```
class SimpleAudioEngine
{
public:
    //获取SimpleAudioEngine的实例
    static SimpleAudioEngine *sharedEngine();

    //注销SimpleAudioEngine的实例
    static void end();

    //设置资源文件
    static void setResource(const char *pszZipFileName);

    //预加载背景音乐
    void preloadBackgroundMusic(const char *pszFilePath);

    //播放背景音乐
    void playBackgroundMusic(const char *pszFilePath, bool bLoop = false);

    //停止播放背景音乐
    void stopBackgroundMusic(bool bReleaseData = false);

    //暂停播放声音
    void pauseBackgroundMusic();

    //恢复播放声音
    void resumeBackgroundMusic();

    //判断是否正在播放背景音乐
    bool isBackgroundMusicPlaying();

    //获取背景音乐的音量大小
    float getBackgroundMusicVolume();

    //设置背景音乐的音量大小
    void setBackgroundMusicVolume(float volume);

    //获取音效的音量大小
```

```
    float getEffectsVolume();

    //设置音效的音量大小
    void setEffectsVolume(float volume);

    //播放音效
    unsigned int playEffect(const char *pszFilePath);

    //停止播放音效
    void stopEffect(unsigned int nSoundId);

    //预加载音效
    void preloadEffect(const char *pszFilePath);

    //卸载音效
    void unloadEffect(const char *pszFilePath);
};
```

现在我们看看在游戏中是如何使用这些函数的，下面是典型的用法：

```
//获取SimpleAudioEngine的实例
SimpleAudioEngine s_SharedEngine = SimpleAudioEngine::sharedEngine();

//预加载音效
s_SharedEngine->preloadEffect("Audio/Open1.ogg");

//播放背景音乐
s_SharedEngine->playBackgroundMusic("Audio/backgrounp.mp3",true);

//停止播放背景音乐
s_SharedEngine->stopBackgroundMusic(true);

//播放音效
int nSoundId = s_SharedEngine->playEffect("Audio/Open1.ogg");

//停止播放音效
s_SharedEngine->stopEffect(nSoundId);

//注销SimpleAudioEngine的实例
s_SharedEngine->end();
```

好了，有了声音模块的支持，大家就可以在游戏内播放声音了，从而，游戏就进入有声世界了。

6.4　总结

本章对cocos2d-x引擎中的一些高级特性做详细的介绍，主要包括物理引擎、粒子系统和声音模块。在接下来的一章里，主要介绍cocos2d-x引擎的周边工具以及交叉编译等。

第7章

cocos2d-x之周边工具

通过学习第5章和第6章，我们已经了解了cocos2d-x引擎的整体架构和一些高级特性，本章我们以沃Phone系统为例学习cocos2d-x的一些周边工具的使用。这些工具包括沃Phone应用程序打包工具、图片编辑器、瓦片地图编辑工具和粒子系统设计工具。

7.1 沃 Phone 应用程序打包工具

沃Phone应用打包工具（QAppPublisher.exe）可实现对沃Phone的基本应用程序、TCOM组件和主屏幕插件等进行打包。产生的安装包可直接在沃Phone终端上进行安装，也可以直接发布到沃Phone商城。

如果安装了沃Phone的SDK，那么可以在“开始”菜单中选择“所有程序→沃Phone SDK→工具→应用发布工具”找到沃Phone应用打包工具。打开该工具的界面如图7-1所示。该工具是一个典型的Windows应用程序，界面从上到下分别是菜单栏、工具栏、操作区和状态栏。



图7-1　沃Phone应用程序打包工具

沃Phone应用程序打包工具支持“向导模式”和“列表模式”两种操作方式。“向导模式”具有引导性，“列表模式”可进行快速操作，它们之间可以自由切换，不会影响数据的处理。

在菜单中选择“新建”或者在工具栏中选择第一个按钮“新建”，也可以在“开始页面”中选择“新建一个沃Phone应用安装”按钮，进入“向导模式”开始对一个新的沃Phone应用安装包进行配置，如图7-2所示。



图7-2　新建一个沃Phone应用安装

从图7-2中可以看出，“向导模式”有3个步骤需要配置，分别是“软件包设置”、“应用配置”和“添加支持文件”，下面分别详细地介绍这3个步骤。

7.1.1　软件包设置

“软件包设置”是对沃Phone应用安装包基本信息的设置，需要对沃Phone安装包设置软件名称、导入安装包的唯一ID号（软件ID）和设置版本号等，如图7-3所示。在图7-3所示的界面中，带有“*”标志的是必填选项。

(1) **软件名称**。给安装包设置一个名称。支持中英文格式。

(2) **软件ID**。为了确保软件包的唯一性，软件ID由沃Phone应用商城分配，一个软件ID只能被一个安装包使用。软件ID是以XML文件格式保存的，这里只需点击“导入”按钮，选择软件ID文件，导入后就会显示该软件包的唯一ID号。下面就是软件ID的一个示例。

```
<?xml version="1.0"?>
<VERSION name = "1.0">
    <SOFT_ID name="WOPHONE_mt_uphone">
        <APP_ID name = "mt_uphone" type="String" >mt_uphone</APP_ID>
    </SOFT_ID>
</VERSION>
```

(3) **软件类别**。软件类别是给软件包的一个分类，有游戏、娱乐和工具等20个类别可供选择。

(4) **软件属性**。标识软件包的应用属性，必须为C/C++。

(5) **安装位置**。安装位置决定对一些数据文件安放处理，包括系统决定、只安装在内部存储区、优先安装在外部存储区。

(6) **版本号**。给安装分配一个版本号，方便升级维护，如1.01.001。

(7) **版权信息**。对软件包版权的一个申明信息。

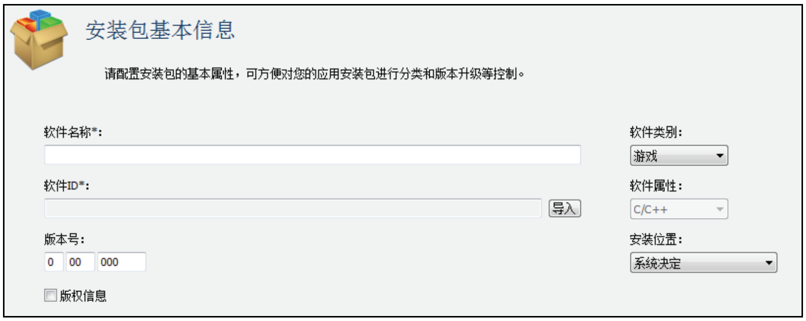


图7-3　软件包设置

将上述信息全部填完之后的一个示例如图7-4所示。



图7-4　软件包设置的示例

配置完“软件包设置”后，可点击向左箭头或者点击“应用配置”按钮，进入“应用配置”页面。

7.1.2 应用配置

在“应用配置”页面，开发者可添加配置沃Phone的应用程序、添加配置应用入口及入口快捷方式。刚进入该页面会提示开发者添加一个应用，如图7-5所示。



图7-5 添加应用

点击“添加”按钮，进入添加应用程序设置界面，如图7-6所示。在该界面中需要选择应用程序文件、选择应用ID及判断是否支持TCOM。

(1) **添加引用程序**。沃Phone应用程序是运行在沃Phone终端上的具体文件。点击“浏览”按钮添加一个沃Phone的应用程序（后缀名为so的库文件）。

(2) **为应用选择一个APP ID**。选择应用ID（APP ID），应用ID是在“软件包设置”里面导入的。这里只需选择一个应用ID，不可以修改。

(3) **支持TCOM**。选择该应用是否支持TCOM。

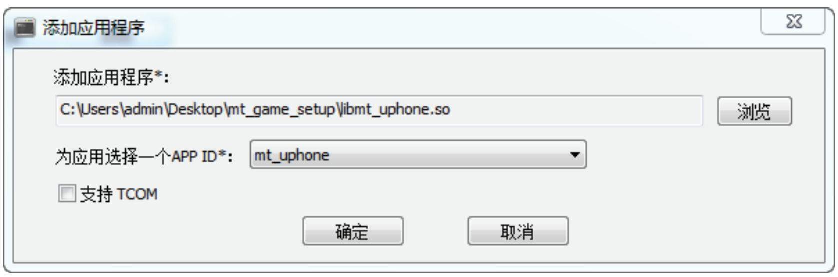


图7-6 添加应用

点击“确定”完成应用的配置添加。当应用添加完成后，回到“应用配置”页面，如图7-7所示。

再次点击“添加”按钮时弹出一个菜单选项，在弹出的菜单选项内可以选择“添加应用”或者“添加入口”。此时，可以点击“添加入口>>”按钮，弹出添加入口界面，如图7-8所示。



图7-7　应用添加完成

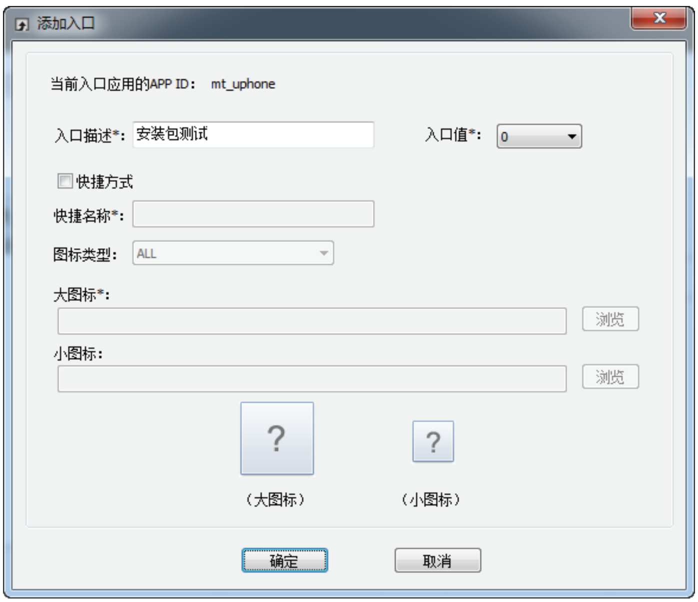


图7-8　为应用添加入口

(1) **当前入口应用的APP ID**。显示了该入口所对应的应用ID（APP ID）。

(2) **入口描述**。入口的描述信息，自动关联了软件包名称，可以自行修改。

(3) **入口值**。可选择范围是0～9的任一个数字，同一个应用多入口时不能设置相同的入口值。

(4) **快捷方式**。选中时，可以将当前入口添加到沃Phone终端主屏幕上的一个图标，如图7-9所示。当选中“快捷方式”时，快捷名称关联着入口描述，可以自行修改，快捷名称会显示在沃Phone终端上。图标类型包括ALL、HVGA和WVGA三种方式。

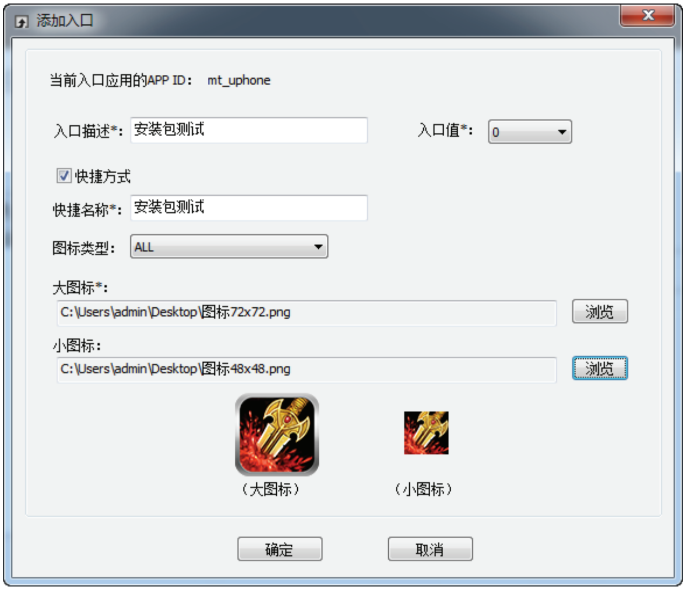


图7-9　为应用添加入口的示例

当对“添加入口”配置完后，点击“确定”按钮返回“应用配置”页面。进入“应用配置”页面后刚刚设置的入口和快捷方式的信息都会展现出来，如图7-10所示。



图7-10　入口设置完成

在树形列表中点击右键选中入口节点，弹出入口控制菜单可对入口进行添加、修改和移除操作。“移除”按钮可移除当前所选的应用或者入口信息。

当对“应用配置”配置完后，可点击向左箭头或者点击“添加支持文件”按钮，进入“添加支持文件”页面。

7.1.3　添加支持文件

进入“添加支持文件”页面，如图7-11所示，有“资源库”、“公共库”、“资料”、“本机”和“SD卡”可供选择添加支持文件。



图7-11　添加支持文件

(1) **资源库**。“资源库”是指应用程序自带的资源包库文件，以动态库形式供应用程序使用。例如不同分辨率的资源包和不同语言的资源包等。

(2) **公共库**。“公共库”是指应用程序自带的公共库文件，以动态库形式供应用程序使用。例如一些公共算法动态库等。

(3) **资料**。“资料”是指存放应用程序使用的不可修改的数据文件根目录，可以在根目录放入一些应用程序需要用到的资料文件，例如图片和声音等文件，也可以添加子目录。

(4) **本机**。“本机”是指手机提供一个用户可见的数据区，可以通过文件管理器查看和修改文件，也可当U盘使用。

(5) **SD卡**。“SD卡”通常指手机的外部存储设备，包括SD卡和T卡等目录，一般一个手机只有一个SD卡存储区。

点击“添加”按钮，弹出选择菜单，可以添加单个文件目录和文件。右键选中列表目录中的列表节点，弹出目录文件控制菜单，“目录文件控制菜单”支持“添加文件”、“添加目录”、“修改名称”、“创建目录”和“移除目录”选项。右边的列表显示了文件的基本信息，选中文件可对其进行移除操作。

完成“软件包设置”、“应用配置”和“添加支持文件”后，就可以对配置的信息进行编译了。

7.1.4　保存编译

在菜单栏上选择“编辑→编译工程”执行编译工作。如果没有保存当前打包的配置文件，会

提示是否设置保存路径，如果选择“是”，则弹出生成文件路径保存设置，如图7-12所示；如果选择“否”，则取消保存，也取消编译。

从图7-12可以看出：执行文件是以tgr为后缀的文件，这个文件可以直接在沃Phone终端上安装；配置文件是以upiproj为后缀的文件，它保存了整个打包的配置信息。

编译成功后会提示制作成功消息框，如图7-13所示；编译失败则会弹出失败消息框。

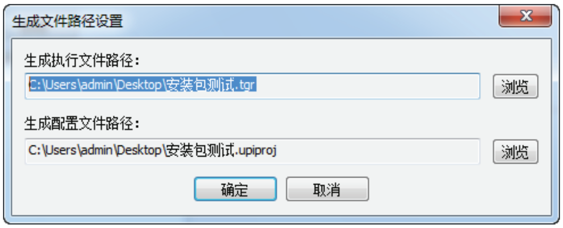


图7-12　生成文件路径设置

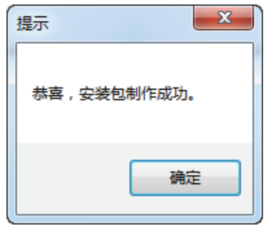


图7-13　编译成功

7.2　图片编辑器

图片编辑器的功能比较简单，就是把一系列的图片拼接成一张大的图片，同时生成一个图片坐标文件。为何要这样呢？这主要是因为引擎在加载图片文件的时候，加载一张图片比加载N张图片的运算量要小很多，占用内存也少很多。

拼接生成的大图片文件和其他图片文件一样，支持PNG等多种格式。至于图片坐标文件，建议使用plist结构，因为cocos2d-x引擎默认支持这个结构，当然也可以写成自定义结构的文件，然后自己解析。

从第5章的内容可以知道，在cocos2d-x引擎里面加载一个图片的代码如下：

```
//加载图片left_bottom.png
CCSprite *left_bottom = CCSprite::spriteWithFile("left_bottom.png");
```

在游戏设计的时候，大部分动画都是预先渲染好的位图，然后通过快速的变化来给玩家一种动态的感觉。现在假设我们有一个Flash的动画，由于cocos2d-x引擎不能直接播放Flash动画，所以需要将这个动画导出成一系列的PNG图片。

Adobe的Flash提供了将帧动画导出为PNG图片序列的功能。根据动画的时间长短，导出的PNG图片个数也不相等，有可能导出几个PNG图片，也有可能导出几十个PNG图片，甚至有可能是几百个PNG图片。此时，尽管我们仍然可以用上面的CCSprite加载这些图片，然后播放动画时分别调用CCSprite，但是这样做会导致游戏占用很多内存，使游戏加载速度变慢，同时也会非常消耗性能。

所以，我们建议使用cocos2d-x引擎的CCSpriteBatchNode这个方法读取拼接后的大图片。得到CCSpriteBatchNode之后，我们可以通过坐标位置直接获取CCSprite，就不需要加载多

次图片了，大致的代码如下。

```
//将图片加载成CCSpriteBatchNode
CCSpriteBatchNode *animationsSheet = CCSpriteBatchNode::batchNodeWithFile("Actor.png");

//从CCSpriteBatchNode获取得到CCSprite
CCSprite  *heroSprite  =  (CCSprite  *)CCSprite::spriteWithTexture(animationsSheet->
getTexture(),CCRectMake(0,0,32,32));
```

上面这个是可以通过CCSpriteBatchNode得到CCSprite的案例，游戏程序中需要经常计算坐标。其实还可以使用前面提到的图片坐标文件（plist文件）保存图片的坐标，以后就可以通过小图片的名称来读取图片了。当然也是一次读取后通过plist文件内的图片名称来获取得到对应的CCSprite，大致的代码如下。

```
//通过plist加载图片
CCSpriteFrameCache *cache = CCSpriteFrameCache::sharedSpriteFrameCache();
cache->addSpriteFramesWithFile("animations/grossini.plist");

//从CCSpriteFrameCache中获取CCSpriteFrame
const char *keyName = "TestAAA";
CCSpriteFrame *frame = cache->spriteFrameByName(keyName);
```

plist文件的生成工具主要包括：

(1) zwoptex。主要包括两个版本，一个是Flash版本（已经不再维护），另一个是Mac版本，收费软件，7天免费试用。

(2) TexturePacker。免费版中会自动向导出的图片中加入红色。TexturePacker拥有zwoptex的90%的功能。

7.3 地图编辑工具

在制作游戏的时候，游戏的地图是游戏的基础，也是游戏的核心之一。游戏地图大致有横向地图和纵向地图，还有一种就是横向和纵向集合的地图。前面说的《魔域之城》这类RPG游戏就是典型的横向和纵向集合的地图。

那么，如何在游戏中快速地显示地图？而且制作地图的过程不影响游戏的开发？解决上述这两个问题就需要使用地图编辑器了。游戏的策划和美工等人员使用地图编辑器绘制地图，而游戏的开发人员根据地图编辑器的结果编写游戏，双方通过地图编辑器进行很好的分工。

如果对cocos2d-x引擎非常熟悉就会发现：cocos2d-x引擎内自带了CCTileMapAtlas、CCTMX-TiledMap、CCTMXXMLParser、CCTMXLayer和CCTMXObjectGroup这几个类，也就是说cocos2d-x引擎直接支持瓦片地图（CCTileMapAtlas），直接支持TMX格式的瓦片地图（CCTMXTiledMap）。

瓦片地图（Tile Map）类似于拼图游戏，又或者像搭积木。元素是可以重复利用的，元素之间相互组合就可以形成一个大场景。图片一般保存在一个图片集中，通过某种映射方式把一个个小格

子映射到相应的图片就形成地图了。cocos2d-x引擎中CCTileMapAtlas的类关系如图7-14所示。

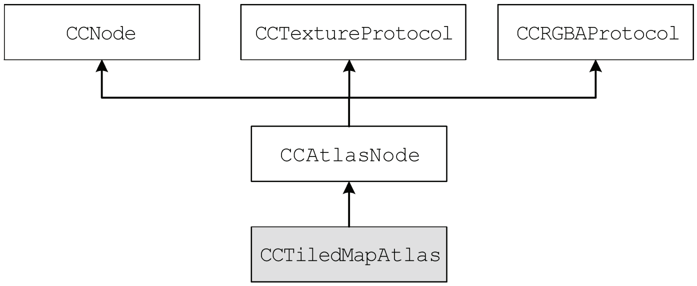


图7-14　CCTileMapAtlas在cocos2d-x引擎中的类结构图

从CCTileMapAtlas的初始化函数中，可以看出CCTileMapAtlas需要使用一个图片和一个TGA文件进行初始化。TGA文件是一种图片文件格式，特点是按行存储且格式简单，图像数据可以压缩也可以不压缩。初始化的示例代码如下：

```
//TileMap资源
const char s_TilesPng[] = "TileMaps/tiles.png";
const char s_LevelMapTga[] = "TileMaps/levelmap.tga";

//TileMap宽度、高度
int i_tileWidth = 16;
int i_tileHeight = 16;

//创建瓦片地图图片集
CCTileMapAtlas *map = CCTileMapAtlas::tileMapAtlasWithTileFile(s_TilesPng, s_LevelMapTga,
i_tileWidth, i_tileHeight);
```

现在，我们再来看看TMX格式的瓦片地图。TMX是一种XML格式，它可以通过一些地图编辑器（比如Tiled QT）生成。在TMX文件里，用XML描述了地图需要用到的图片、地图的布局以及地图的层等各种信息。首先，我们来了解一下TMX里面的常用概念，即瓦片（Tile）、瓦片集（Tile Set）、瓦片ID（GID）、层（Layer）、对象（Object）、对象组（Object Group）、属性（Property）和地图朝向（Orientation）。

- **瓦片**。一个瓦片对应于一个矩形区域，你可以把某块地方分割成3行3列，那就是有9个瓦片，每个瓦片里面放上图片，就形成地图。
- **瓦片集**。瓦片的图片一般都是保存在一个图片集（Atlas）中的，而我们这里说的是瓦片集，和图片集是两个概念。图片集是把单个图片组合起来，瓦片集也就是把瓦片打包起来进行管理。
- **瓦片ID**。瓦片ID 可以标识某个网格里显示什么图片。某一个地方被分割成3行3列格子，现在设置第1行第1列的瓦片ID为5。而该地图所使用的图片集是一个包含有16×16的地图图片，由于5除以16等于0，余数为5，所以也就是说明第1行第1列的这个格子应该显示图片集中第1行第5列的图片。

- **层**。TMX地图支持分层，类似于PhotoShop中层的概念，可以把地图中的元素放到不同的层中。如果不想显示某一层，可以很方便地隐藏掉，增加了灵活性。地图的层可以大小不一样，层也可以指定一个偏移值，并不一定非得原点都对齐。
- **对象**。对象是为了开发者方便而设计的，它并不对应于某个地图图片，只是标明了某个位置，可以通过对象的名字获取对象。
- **对象组**。把多个对象包起来进行分组，可以通过对象组的名字获取对象组。
- **属性**。TMX里面的元素基本都可以设置额外属性，可以通过属性的名字获取属性。
- **地图朝向**。TMX文件里可以指定地图的朝向，目前支持三种，即正交型、等角型和六边形。

众所周知，Tiled是一个开源的TMX编辑工具。目前，Tiled一共有两个版本，一个是Java版本，另一个是QT版本，随便选择一个都可以。图7-15就是Tiled的QT版本打开时的界面。

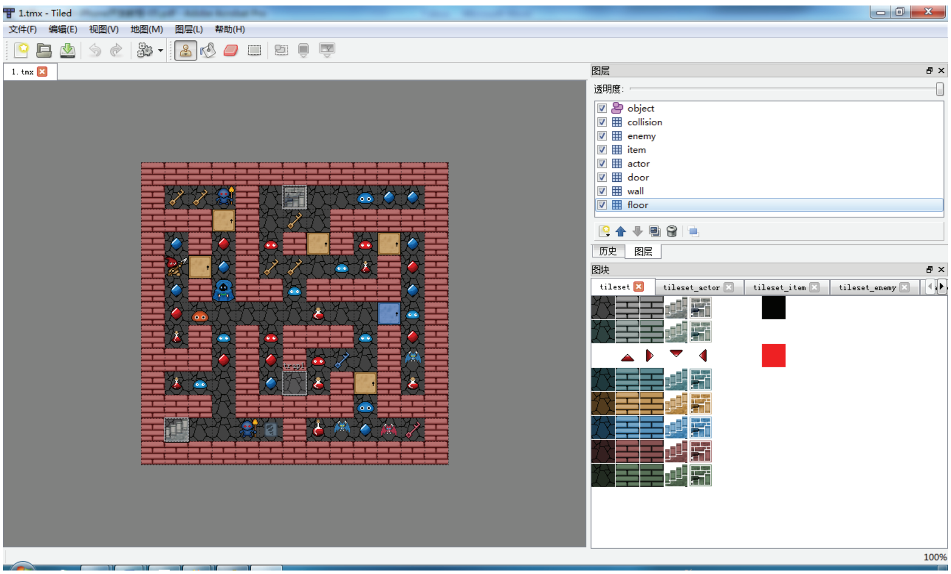


图7-15　Tiled的QT版本打开时的界面

在cocos2d-x引擎里面，CCTMXTiledMap（TMX瓦片地图）主要的函数解释如下（在文件include/ CCTMXTiledMap.h里面）：

```
class CCTMXTiledMap : public CCNode
{
    //使用TMX文件创建CCTMXTiledMap
    CCTMXTiledMap *tiledMapWithTMXFile(const char *tmxFile);

    //使用TMX文件初始化CCTMXTiledMap
    bool initWithTMXFile(const char *tmxFile);
```

```
    //通过层的名称获取层
    CCTMXLayer *layerNamed(const char *layerName);

    //通过对象组的名称获取对象组
    CCTMXObjectGroup *objectGroupNamed(const char *groupName);

    //通过对象组的名称获取对象组
    CCTMXObjectGroup *groupNamed(const char *groupName);

    //通过属性的名称获取属性
    CCString *propertyNamed(const char *propertyName);
};
```

使用Tiled设计完地图之后，将其导出成TMX格式的文件，这个地图文件在cocos2d-x引擎中可以直接使用，创建TMX瓦片地图的代码如下：

```
//创建TMX瓦片地图，地图文件orthogonal-test3.tmx
CCTMXTiledMap *map = CCTMXTiledMap::tiledMapWithTMXFile("orthogonal-test3.tmx");
```

7.4 粒子系统设计工具

在前面几章，我们已经学习了cocos2d-x引擎中粒子系统的使用。为了实现那些粒子效果，我们在程序中需要写很多的参数。那么，有没有一个工具可以帮我们设计并生成粒子系统需要的这些参数呢？答案是：有的。现在我们就来看看Particle Designer这个粒子系统设计工具。

Particle Designer是cocos2d-x引擎支持的粒子系统设计工具，该软件只支持Mac OS平台，并且是收费软件，不过软件质量确实不错。Particle Designer提供了试用版本，但试用版本不能保存粒子效果为plist文件。图7-16就是Particle Designer软件界面。

当使用Particle Designer设计完成粒子效果之后，可以将其保存为plist文件。plist文件的内容是一个XML结构，其中部分内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyListvb-1.0.dtd">
<plist version="1.0">
<dict>
    <key>angle</key>
    <real>90</real>
    <key>angleVariance</key>
    <real>360</real>
    <key>blendAdditive</key>
    <real>1</real>
    <key>blendFuncDestination</key>
    <integer>1</integer>
    <key>blendFuncSource</key>
    <integer>770</integer>
    <key>duration</key>
    <real>-1</real>
```

```
        ......
        ......
        ......
        ......
        <key>startColorVarianceRed</key>
        <real>0.0</real>
        <key>startParticleSize</key>
        <real>37</real>
        <key>startParticleSizeVariance</key>
        <real>10</real>
        <key>tangentialAccelVariance</key>
        <real>50</real>
        <key>tangentialAcceleration</key>
        <real>50</real>
        <key>textureFileName</key>
        <string>Galaxy.png</string>
        <key>textureImageData</key>
    </dict>
    </plist>
```

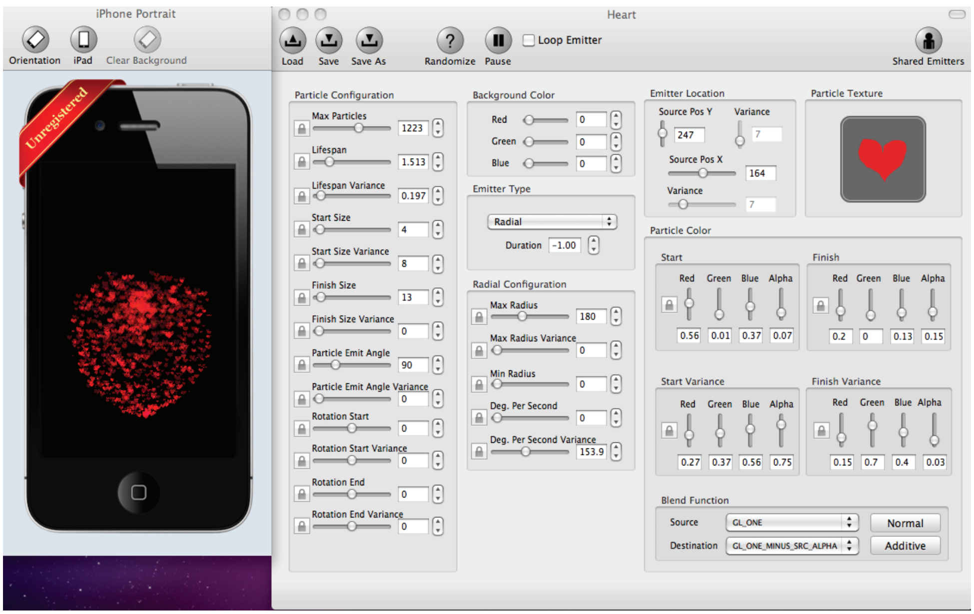


图7-16 Particle Designer软件界面

从上面plist的部分内容可以看出，plist文件就是将粒子系统所需要的参数以XML形式组装起来。而由于cocos2d-x引擎直接就支持plist文件作为粒子系统的参数，所以我们的代码就可以重新精简成下面这个样子：

```
//建立一个Quad的粒子系统
CCParticleSystem *m_emitter = new CCParticleSystemQuad();
```

```
//使用plist文件初始化粒子系统
m_emitter->initWithFile("Images/SpinningPeas.plist");

//设置粒子的图像
m_emitter->setTexture( CCTextureCache::sharedTextureCache()->addImage("test.png") );
```

cocos2d-x引擎不仅支持Particle Designer的所有特性，而且还提供了一些Particle Designer不支持的特性。

当使用重力式的粒子系统时，相比Particle Designer，cocos2d-x引擎增加了4个属性：tangentialAccel（切向加速度）、tangentialAccelVar（切向加速度的变化率）、radialAccel（径向加速度）和radialAccelVar（径向加速度的变化率）。在上一章的粒子系统示例代码中能看到这4个属性的使用方式。代码示例如下：

```
//设置径向加速度、径向加速度的变化率
m_emitter->setRadialAccel(-120);
m_emitter->setRadialAccelVar(0);

//设置切向加速度、切向加速度的变化率
m_emitter->setTangentialAccel(30);
m_emitter->setTangentialAccelVar(0);
```

当使用放射式的粒子系统时，相比Particle Designer，cocos2d-x引擎增加了4个属性：startSpin（粒子开始时自旋转速度）、startSpinVar（粒子开始时自旋转速度的变化率）、endSpin（粒子结束时自旋转速度）和endSpinVar（粒子结束时自旋转速度的变化率）。在上一章的粒子系统示例代码中能看到这4个属性的使用方式。代码示例如下：

```
//设置粒子开始时自旋转速度、粒子开始时自旋转速度的变化率
m_emitter->setStartSpin(0);
m_emitter->setStartSpinVar(0);

//设置粒子结束时自旋转速度、粒子结束时自旋转速度的变化率
m_emitter->setEndSpin(0);
m_emitter->setEndSpinVar(2000);
```

当使用放射式的粒子系统时，Particle Designer支持下面几个属性：maxRadius（最大半径）、maxRadiusVariance（最大半径的变化率）和minRaidus（最小半径）。而cocos2d-x引擎比Particle Designer增加了更多几个属性：startRadius（开始时半径）、startRadiusVar（开始时半径的变化率）、endRadius（结束时半径）和endRadiusVar（结束时半径的变化率）。

7.5　总结

为了方便游戏的开发，游戏的开发者需要熟悉引擎周边的很多工具的使用，本章我们介绍了沃Phone应用程序的打包工具、图片编辑器、瓦片地图编辑器和粒子系统设计工具。当然围绕着cocos2d-x引擎周边还有很多工具，需要大家在开发游戏时慢慢摸索。

在下一章里，我们将学习如何把开发好的游戏交叉编译到iOS、Android和沃Phone这三个平台上。

第8章

cocos2d-x之交叉编译

在前面几章，我们学习了很多cocos2d-x引擎的知识，我们知道cocos2d-x引擎是一个跨平台的游戏引擎，那么，使用cocos2d-x引擎开发出来的游戏也应该是一个跨平台的游戏。游戏只需要做一点点的变动就可以交叉编译到不同的平台上，而且这一点点的变动是固定的。

那么，本章就看看如何将编写好的游戏交叉编译到iOS、Android和沃Phone平台上。

8.1 交叉编译到 iOS 平台

编译在iOS上运行的游戏，至少需要一个iOS的项目，然后使用iOS开发环境Xcode提供的编译工具进行交叉编译。因此将游戏交叉编译到iOS平台上主要包括新建iOS项目、交叉编译和打包运行三步。

8.1.1 新建iOS项目

在Mac操作系统上运行Xcode3，创建一个基于“cocos2d-x Application”模板的iOS项目（如图8-1所示），项目名称为TestBuildGame。

点击“choose”（选择）按钮，TestBuildGame的iOS项目创建完成。此时，我们可以看到项目的结构，如图8-2所示。

现在TestBuildGame中的代码都是自动生成的代码，显然不满足要求，所以，我们将TestBuildGame项目的Resource目录和Classes目录中的内容删掉，然后将我们自己开发的游戏（此处举例为HelloWorld游戏）的Resource目录和Classes目录中的内容复制到TestBuildGame项目中。

在iOS开发中，经常需要使用一些资源文件（如视频、音频和图片等）。但在默认情况下iOS系统会将所有的资源全部复制到mainBundle（全部放置在一个文件夹下）目录下，即使资源是按文件夹来组织的。

这种处理方式很多时候并不满足需求，程序经常需要按照文件夹的方式管理资源。所以我们需要解决这种问题，其实解决这个问题也很简单，当资源文件夹拖到Xcode工程的“Other sources”目录下时，Xcode会弹出一个对话框（如图8-3所示），设置资源的访问方式，默认的选项是“Recursively create groups for any added folders”,此时我们选择下面的一项“Create Folder

References for any added folders”，然后选择“Add”（添加）按钮。此后在代码中就可以用文件夹的路径方式访问资源了。

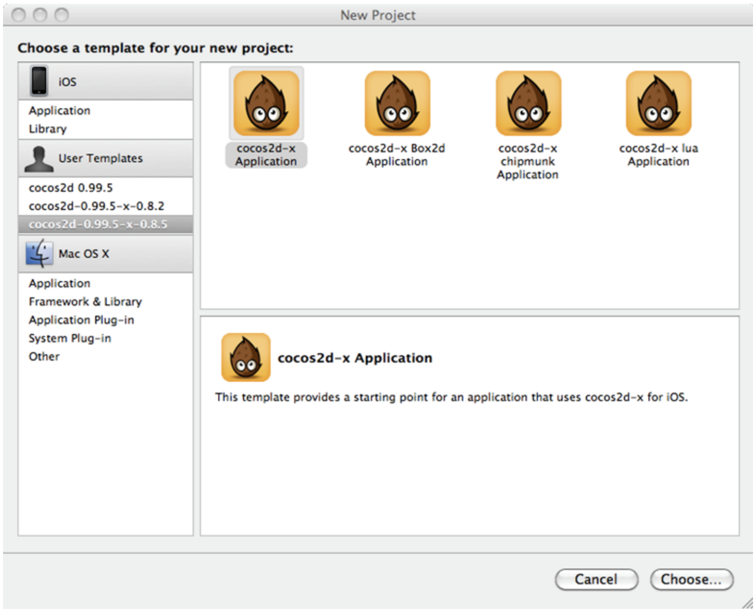


图8-1　新建iOS项目

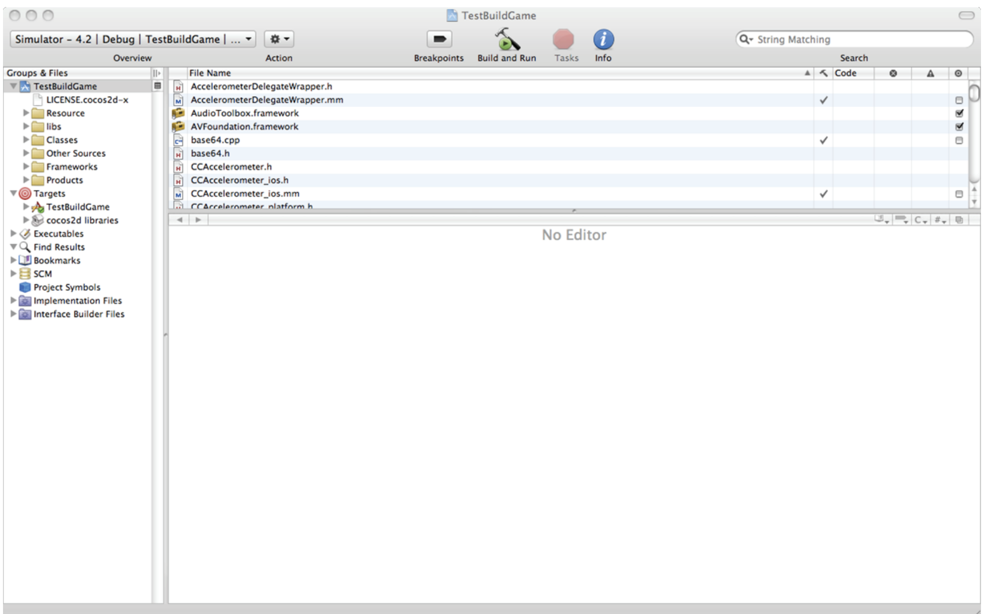


图8-2　TestBuildGame的iOS项目结构

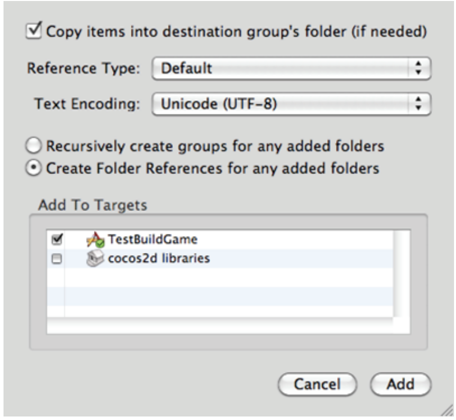


图8-3　设置资源访问方式

8.1.2　交叉编译

前面几步已经完成进行iOS项目交叉编译的所有前提准备了，那么现在就要正式开工了。直接使用Xcode的“Build”按钮即可编译完成。

8.1.3　打包运行

执行完交叉编译之后，我们就可以在iOS机器上运行游戏了。在iOS仿真器上最终运行的效果如图8-4所示。



图8-4　运行效果

8.2　交叉编译到Android平台

刚才介绍了如何把游戏交叉编译到iOS平台上，现在来学习如何把Hello World游戏编译到Android平台上。

编写过Android游戏的人都应该知道：编译在Android上运行的游戏至少需要一个Android的项目，然后使用Android的SDK和NDK提供的工具进行交叉编译。因此将游戏交叉编译到Android平台上主要包括新建Android项目、生成编译脚本、交叉编译和打包运行四步。

8.2.1　新建Android项目

我们需要使用安装了Android开发工具的Eclipse新建一个空的Android项目，项目名称为TestBuild -Game，包的名称为org.test.build.game，支持Android的SDK版本为2.1。生成的项目结构如图8-5所示。

现在TestBuildGame中的代码都是自动生成的代码，显然不满足要求。我们还需要完成下面几个步骤的工作：

(1) 将例子代码的src目录复制到TestBuildGame项目中；

(2) 将例子代码的libs目录复制到TestBuildGame项目中；

(3) 将例子代码的jni目录复制到TestBuildGame项目中；

(4) 将例子代码的build_native.sh文件复制到TestBuildGame项目中。

全部导入代码后生成的项目结构如图8-6所示。

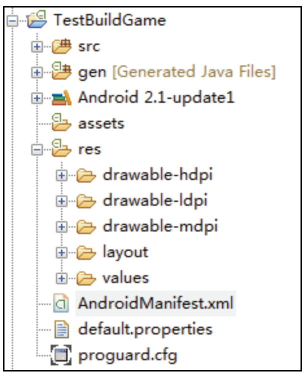


图8-5　TestBuildGame的Android项目结构

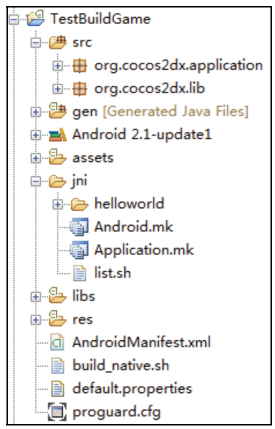


图8-6　导入配置后的Android项目

8.2.2　生成编译脚本

细心的读者一定有疑问，刚才在复制文件的时候为什么没有复制使用cocos2d-x引擎开发的游戏代码呢？是不是作者忘记写了呢？其实不是的，因为Android的项目需要使用Android的NDK进行编译，而Android的SDK对代码的位置没有要求，只要求生成的Linux库文件（so文件）必须放在项目的libs/armeabi目录中。

我们在复制文件的时候复制了build_native.sh文件，这个文件就是用来编译我们的游戏代码

的，下面就是一个示例：

```
# Android的NDK路径
ANDROID_NDK_ROOT=/cygdrive/c/android-ndk-r5b

# cocos2d-x项目的路径
COCOS2DX_ROOT=/cygdrive/c/cocos2d-0.99.5-x-0.8.5

# TestBuildGame项目的路径
TESTBUILDGAME_ROOT=$COCOS2DX_ROOT/HelloWorld/android

# 在项目中建立assets目录
if [ -d $TESTBUILDGAME_ROOT/assets ]; then
    rm -rf $TESTBUILDGAME_ROOT/assets
fi
mkdir $TESTBUILDGAME_ROOT/assets

# 复制Resource目录中的文件
for file in $TESTBUILDGAME_ROOT/HelloWorld/Resource/*
do
    if [ -d $file ]; then
        cp -rf $file $TESTBUILDGAME_ROOT/assets
    fi

    if [ -f $file ]; then
        cp $file $TESTBUILDGAME_ROOT/assets
    fi
done

# 编译
$ANDROID_NDK_ROOT/ndk-build -C $ TESTBUILDGAME_ROOT $*
```

接下来我们需要再去修改项目jni/helloworld目录中的Android.mk文件，这个文件指定了当前Android项目在NDK编译时的参数。那我们就来仔细研究一下这个Android.mk的结构，主要参数见表8-1。

表8-1 Android.mk的主要参数

参数名称	参数描述
LOCAL_PATH	项目的根路径
LOCAL_MODULE	项目的输出文件名
LOCAL_SRC_FILES	包含C、C++文件的名字
LOCAL_C_INCLUDES	包含路径
LOCAL_LDLIBS	连接的库文件

现在我们来看一下TestBuildGame项目jni/helloworld目录中的Android.mk文件，内容如下。

```
LOCAL_PATH :=$(call my-dir)
```

```
include $(CLEAR_VARS)
LOCAL_MODULE :=TestBuildGame

LOCAL_SRC_FILES :=main.cpp \
../../../AppDelegate.cpp \
../../../HelloWorldScene.cpp

LOCAL_C_INCLUDES :=$(LOCAL_PATH)/../../../../cocos2dx \
                   $(LOCAL_PATH)/../../../../cocos2dx/platform \
                   $(LOCAL_PATH)/../../../../cocos2dx/include \
                   $(LOCAL_PATH)/../../../../cocos2dx/lua_support \
                   $(LOCAL_PATH)/../../../../CocoDenshion/include \
                   $(LOCAL_PATH)/../../..

LOCAL_LDLIBS :=-L$(call host-path, $(LOCAL_PATH)/../../libs/armeabi)\
                  -lcocos2d-llog-lcocosdenshion \
               -L$(call   host-path,   $(LOCAL_PATH)/../../../../cocos2dx/platform/
                  third_party/android/libraries)-lcurl
include $(BUILD_SHARED_LIBRARY)
```

有了上面这几个配置文件，我们就可以开始交叉编译项目了。

8.2.3　交叉编译

前面几步已经完成进行Android项目交叉编译的所有前提准备了，那么现在就要正式开工了。首先运行交叉编译的环境，启动文件是C:\cygwin\ Cygwin.bat。

在这个环境下，我们就可以通过刚才的build_native.sh文件进行交叉编译了。进入D:\Work7\cocos2d-0.99.5-x-0.8.5\TestBuildGame目录，然后运行其中的build_native.sh文件，如图8-7所示。

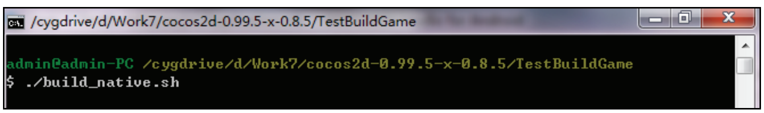


图8-7　编译TestBuildGame的Android项目

TestBuildGame的Android项目编译完成之后，在Eclipse环境中刷新项目，此时TestBuildGame的assets目录中会自动复制我们开发的游戏的Resource目录中的内容，而libs/armeabi目录中则复制了编译出来的Linux库文件（so文件）。

由于Eclipse具有自动编译功能，因此TestBuildGame的Android执行文件（apk文件）已经生成了，文件存放的路径是D:\Work7\cocos2d-0.99.5-x-0.8.5\TestBuildGame\bin。

8.2.4　打包运行

执行完交叉编译之后，我们就得到了Android的可执行文件TestBuildGame.apk，然后将这个文件部署到真机器上就可以运行了。在Android模拟上最终运行的效果如图8-8所示。



图8-8 运行效果

8.3 交叉编译到沃 Phone 平台

前面已经介绍了把游戏交叉编译到iOS平台和Android平台上，现在我们来学习如何把HelloWorld游戏编译到沃Phone平台上。

编译在沃Phone上运行的游戏，至少需要一个沃Phone的项目，这样才能使用沃Phone的SDK提供的工具进行交叉编译。因此将游戏交叉编译到沃Phone平台上主要包括新建沃Phone项目、生成编译脚本、交叉编译和打包运行四步。

8.3.1 新建沃Phone项目

使用Visual Studio 2008打开cocos2d-x引擎的沃Phone项目，项目解决方案文件是D:\Work7\cocos2d-0.99.5-x-0.8.5\cocos2d-wophone.sln。

创建一个基于“Cocos2d-wophone Application”模板的沃Phone项目（如图8-9所示），项目名称为TestBuildGame，项目的保存位置必须在D:\Work7中。

点击“确定”按钮，界面变成添加项目的具体信息。在建立项目的过程中，会遇到图8-10所示的界面，这里是选择项目是否支持物理引擎（Box2D）、物理引擎（Chipmunk）、声音引擎（CocosDenshion）或TCOM支持的特性，这里需要按照具体的项目要求选择。在我们的示例中，这几个选项都没有被选中，也就是项目示例不需要这几个特性。

点击“Finish”（完成）按钮，TestBuildGame的沃Phone项目创建完成。此时，我们可以看到项目的结构，如图8-11所示。

从图8-11中可以看出这是一个典型的C++的项目，资源在Resource目录中，与沃Phone平台相关的代码在wophone目录中，其余代码在Classes目录中。还包括用于编译的一些脚本Makefile.ARM和TestBuildGame_Arm.TMK3。

现在TestBuildGame中的代码都是自动生成的代码，显然不满足要求，所以，我们将该

TestBuildGame项目的Resource目录和Classes目录中的内容删掉。然后将我们自己开发的游戏（此处举例为Hello World游戏）的Resource目录和Classes目录中的内容复制到TestBuildGame项目中。

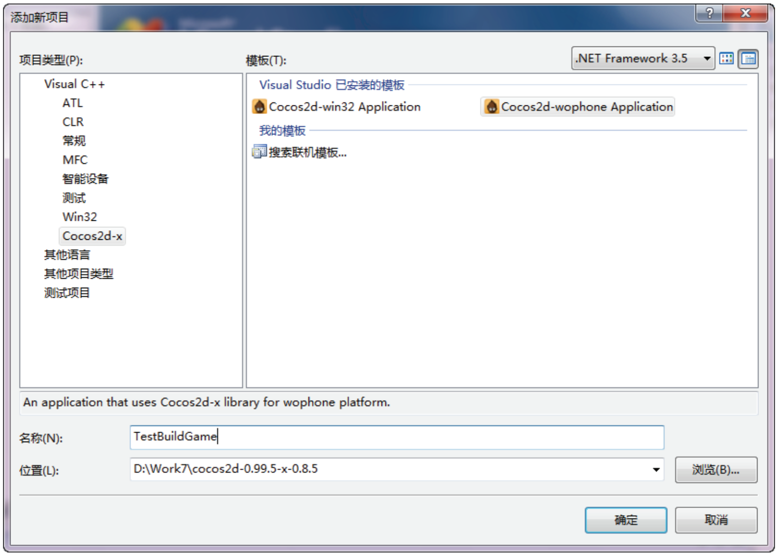


图8-9　新建沃Phone项目

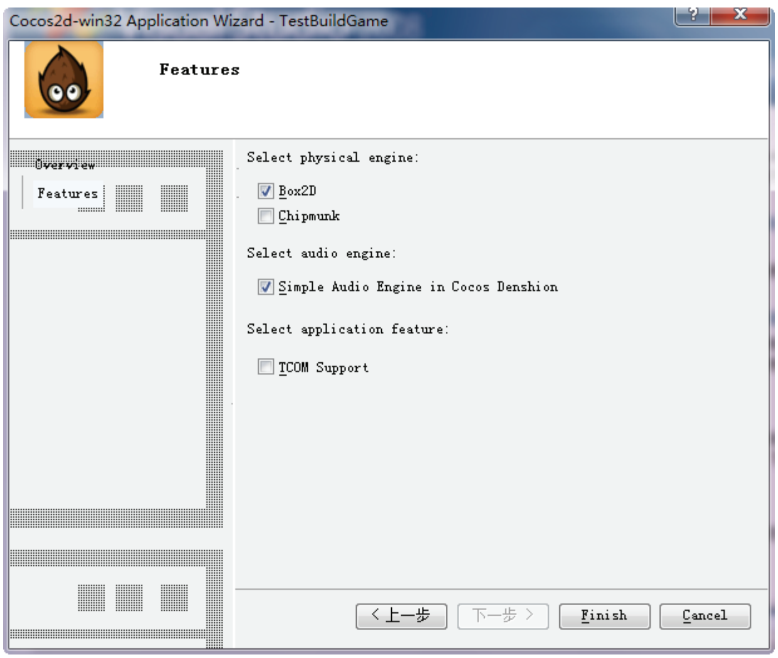


图8-10　添加沃Phone项目时选择特性

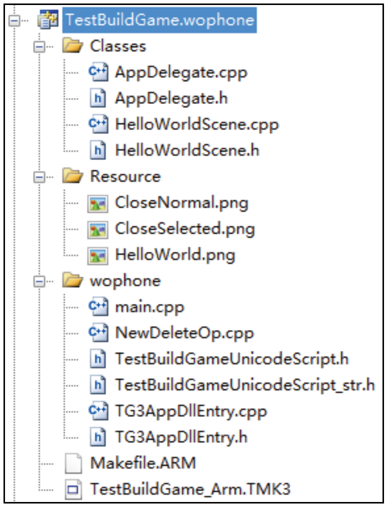


图8-11　TestBuildGame的沃Phone项目结构

此时，还需要将复制过来的这些文件添加到项目中。因为Visual Studio并不会自动将文件夹下的文件设置在项目内，需要手动设置。设置的方式是选中文件后右键选中“包含在项目中”菜单即可。

8.3.2　生成编译脚本

如果大家编写过Linux的程序，一定知道在编写完程序代码后，还需要编写编译脚本makefile。同样，想要编译出沃Phone的程序，也需要一个交叉编译脚本。然后用这个脚本编译出沃Phone平台上的库文件（沃Phone的底层核心是Linux，库文件也是so文件）。

不过，值得庆幸的是，在沃Phone平台上，不需要手动编写makefile文件，因为沃Phone平台提供一个工具，可以自动生成交叉编译脚本。这个工具就是“TMK3文件处理工具”，该工具的路径是D:\Work7\PRJ_TG3\Common\TMK3\TMK3.exe。这个工具是一个Console控制台程序，它需要一个交叉编译参数文件（文件扩展名为TMK3）。现在我们发现，在刚才生成的代码中，有一个TestBuildGame_Arm.TMK3文件。那么我们就来仔细研究一下TestBuildGame_Arm.TMK3的结构。交叉编译参数文件的主要参数见表8-2。

表8-2　交叉编译参数文件的主要参数

参数名称	参数描述
TO_PROJECT_ROOT	项目的根路径
OUTPUT_FILENAME	项目的输出文件名
INCLUDE_TMK3	包含的其他TMK3文件
PRE_DEFINE	预定义串

（续）

参数名称	参数描述
DEFINES	C、C++预定义宏
INCLUDE_PATH	包含路径
LIBS	连接的库文件
INCLUDEFILE	强制包含文件的名字
EXCLUDEFILE	强制排除文件的名字

交叉编译参数文件的整体原则包括：

(1) 在等号左边不要有空格；

(2) 所有的路径请使用“/”来分隔；

(3) 所有的文件名不可以有空格；

(4) 只能对当前目录及其子目录下的c、cpp文件生成交叉编译脚本；

(5) 文件目录位置到项目根目录之间的转换，不支持多个串，以最后一个为准。

现在我们来看一下TestBuildGame这个项目的交叉编译参数文件，内容如下：

```
;
; TG3 Makefile Auto Create Script
;

;项目的根路径
TO_PROJECT_ROOT=../../PRJ_TG3

;项目的输出文件名
OUTPUT_FILENAME=libTestBuildGame.so

;包含的其他的TMK3文件名
INCLUDE_TMK3=$(TO_PROJECT_ROOT)/MakeInclude/TG3_APP_Arm.TMK3

;C、C++预定义宏
DEFINES=-DCC_UNDER_WOPHONE

;包含路径
INCLUDE_PATH=-I  ../../PRJ_TG3/Include/OpenGL  -I../cocos2dx  -I../cocos2dx/include
-I../cocos2dx/platform
INCLUDE_PATH=-I. -I./Classes -I./wophone -I./wophone/Res -I./wophone/Resource

;连接的库文件
LIBS=-lCocos2dStatic -lTG3_EGL   -lTG3_GLESv1_CM -lTG3_GLESv2   -lz -lxml2 -lpng14
-lImageToolKit -ljpeg

;强制包含文件的名字
INCLUDEFILE=
```

```
;强制排除文件的名字
EXCLUDEFILE=
```

上面这个文件就是TestBuildGame项目的交叉编译参数文件。有了这个文件，我们就可以使用TMK3文件处理工具生成交叉编译脚本Makefile.ARM了。生成方式有下面两种。

- 方法一，进入项目目录后，用命令行直接编译TMK3。打开cmd命令行并且进入工程目录，输入命令：D:\Work7\PRJ_TG3\Common\TMK3\TMK3.exe TestBuildGame_Arm.TMK3。
- 方法二，将TMK3文件关联到TMK3.exe，然后直接点击TMK3文件。

使用上述两种方法中的任何一个，我们都可以得到交叉编译脚本Makefile.ARM。有了这个脚本，我们就可以开始编译了。

8.3.3 交叉编译

前面几步已经完成进行沃Phone项目交叉编译的所有前提准备了，那么现在就要正式开工了。首先运行交叉编译的环境，启动文件是D:\Work7\ToolChain\toolchain.bat。

在这个环境下，我们需要先编译cocos2d-x引擎的沃Phone项目。进入D:\Work7\cocos2d-0.99.5-x-0.8.5目录，然后运行其中的build-wophone.sh文件，如图8-12所示。

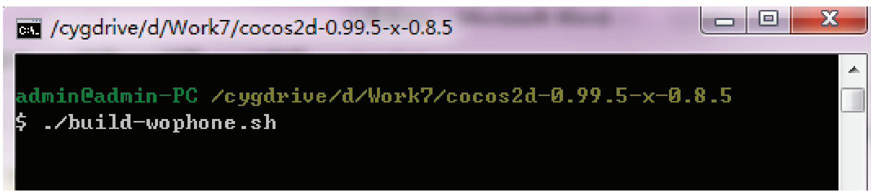


图8-12　编译cocos2d-x引擎的沃Phone项目

在cocos2d-x引擎的沃Phone项目编译完成之后，就要找到我们的项目目录，然后交叉编译生成库文件libTestBuildGame.so，生成方式为make -f Makefile.ARM，如图8-13所示。如果需要清理libTestBuildGame.so可以使用make -f Makefile.ARM clean。

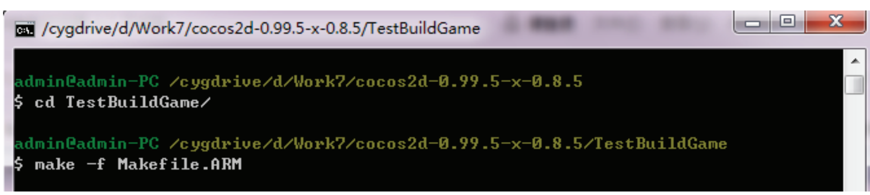


图8-13　编译TestBuildGame项目

使用图8-13所示的方式生成的libTestBuildGame.so比较大，需要通过arm-eabi-strip这个工具给文件“瘦身”，执行的命令行代码为arm-eabi-strip /cygdrive/d/Work7/PRJ_TG3/LIB/ARMLib/libTest-BuildGame.so，如图8-14所示。

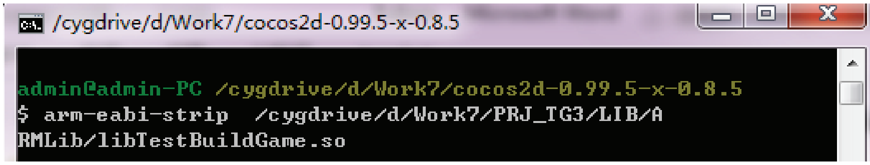


图8-14　给libTestBuildGame“瘦身”

8.3.4　打包运行

执行完交叉编译之后，我们就得到了文件libTestBuildGame.so，然后我们将TestBuildGame项目下的Resource目录压缩成zip文件。

接下来大家就可以通过上一章介绍的“沃Phone应用打包工具”的相关知识打包生成沃Phone的tgr执行文件了，然后将这个文件部署到真机器上就可以运行了。在沃Phone OS仿真器上最终运行的效果如图8-15所示。



图8-15　运行效果

8.4　总结

本章主要介绍了如何将自己编写的游戏进行跨平台的编译发布，这样开发者只需要编写一次代码，然后将代码交叉编译到iOS、Android和沃Phone平台上就行了。在接下来的章节里，我们将学习如何为游戏带来盈利以及游戏的制作全过程。

第9章

cocos2d-x之实用篇

通过前几章的学习，我们不仅掌握了cocos2d-x引擎的相关知识，还学会了使用cocos2d-x引擎的周边工具。估计大家已经开始用前面学习到的知识写游戏了。

其实前面所介绍的知识用来编写单机版的游戏已经足够了，但是如果想要写一个带有社交功能的游戏或者想让游戏嵌入一些广告,又该如何呢？估计大家就不知所措了，因为直到目前还没有介绍服务器相关的知识。

在本章将给大家介绍一种仅仅需要少量代码，就可以把一个单机游戏加上网络化体验的方法，从而给游戏带来可玩性的巨大提升，并可以迅速扩大用户规模和提升用户黏性。这种方法就是让单机游戏支持“游戏社交平台”。

而一旦你的游戏开发完成之后，肯定需要考虑游戏的盈利模式。那么除了在苹果App Store上进行付费下载外，我们还可以怎么盈利呢？本章将给大家介绍如何嵌入广告给游戏带来收入，也就是让游戏嵌入“手机广告平台”提供的广告以达到盈利的目的。然后再给大家带来一种介于游戏社交平台和手机广告平台之间的服务，这就是推广墙平台。推广墙平台的作用是通过虚拟货币和流量交换的方式，获取更多的玩家，并让玩家免费持续地玩下去，同时还能让开发者获得更多的收入。

本章通过在Hello World游戏上增加游戏社交平台、手机广告平台和推广墙平台，帮助大家了解基于cocos2d-x引擎下开发的游戏与游戏社交平台、手机广告平台和推广墙平台集成的相关知识。

9.1 游戏社交平台

一般来讲，游戏社交平台就是一系列的社交功能的模块。像我们将要介绍的微云游戏社交平台,可以提供一系列的社交功能模块,即支持在iOS和Android游戏中方便地嵌入附加的社交功能，比如排行榜、成就、挑战、讨论/反馈、分享和即时消息等。

WiGame是微云提供的游戏社交平台，包括iOS和Android两个版本。我们介绍的cocos2d-x引擎与游戏社交平台的集成就是介绍cocos2d-x引擎与WiGame的集成。通过WiGame，开发者仅仅需要编写少量代码，就可以为一个单机游戏增加网络化的体验，然后通过微云平台发行、运营和推广就可以拥有微云庞大的游戏用户群体。下面就是WiGame提供的模块。

(1) **排行榜系统**。社交游戏可能都需要一个用户排行榜，可以根据游戏的策划设计，通过微

云创建多个不同属性的排行榜。

(2) **成就系统**。设置游戏成就，去激励玩家达成和炫耀！成就可以设置荣誉值，玩家达成成就后便可获得相应的荣誉值或积分。

(3) **挑战系统**。增加异步的挑战，让游戏变得更加有趣，玩家可以互相挑战。

(4) **讨论/反馈系统**。不用再为收集用户反馈苦恼，用户可以从WiGame游戏社区或网页上直接提问，开发者可以进行即时的回复。

(5) **下载链接**。如果开发者已把应用发布在了苹果的App Store或者谷歌的Android Market上面，或者其他任何地方，只要是可以通过URL访问的，那么开发者就可以在管理后台添加应用下载链接，然后在WiGame游戏社区中将可以看到这些下载点，帮助游戏进行强力的推广。

9.2　手机广告平台

游戏社交平台可以让一个单机游戏加上网络化的体验，从而给游戏带来可玩性的巨大提升。而手机广告平台则可以让游戏带来收入。上面介绍了游戏社交平台，现在再来看看手机广告平台。

首先我们了解一下什么是广告。广告是为了某种特定的需要，通过一定形式的媒体，公开而广泛地向公众传递信息的宣传手段。

在没有手机广告平台的时候，一个开发者很难接触到需要发布广告的广告主，即使接触到，由于开发者相对广告主在定价等方面是弱势群体，开发者很难从广告中获取利益。

现在，国内有了一些专门的广告平台，这些广告平台是开发者与广告主之间的桥梁。广告平台统一与广告主谈判沟通，吸引广告主在平台上面发布广告。然后广告平台将广告主发布的广告交给开发者进行展现，按照展现的次数或者点击的次数给开发者付费。

WiAd是微云提供的手机广告平台，包括iOS和Android两个版本。我们介绍的cocos2d-x引擎与手机广告平台的集成就是cocos2d-x引擎与WiAd的集成。通过WiAd，开发者仅仅需要少量代码，就可以给一个游戏嵌入广告，这样就可以带来收入。

WiAd主要包括两大部分，一部分是给广告主使用的，另一部分是给开发者使用的。

(1) 为广告主提供操作简单、专业高效的广告投放平台。提供了创建广告计划、创建广告投放规则、创建广告素材、广告主的广告统计报表和广告主的区域统计报表等模块。通过这些模块，可以形成丰富多样的广告形式，提升广告传播效果。广告主可以选择的5种广告类型见表9-1。

表9-1　广告主可以选择的5种广告类型

英文简写	英文全称	描　　述
CPC	Cost Per Click	点击付费广告，广告主根据广告被有效点击的次数收费
CPM	Cost Per Thousand Impression	展示付费广告，只要展示了广告主的广告内容，广告主就为此付费

（续）

英文简写	英文全称	描　述
CPA	Cost Per Action	一种按广告投放实际效果计价方式的广告，即按回应的有效行为来计费
CPS	Cost Per Sales	一种以实际销售产品数量来计算广告费用的广告
CPT	Cost Per Time	一种以时间来计费的广告，国内很多的网站都是按照“一个月多少钱”这种固定收费模式来收费的

(2) 为开发者提供嵌入广告的方式和稳定的应用管理平台。开发者可以随时提交应用，查看相应数据，管理应用程序。开发者通过WiAd的SDK可以灵活控制广告显示的次数、频率和类型等方面设置。WiAd的SDK可以进行广告的展示，并自动跟踪、统计广告的展示和点击。同时，WiAd的SDK提供了丰富的编程接口对广告显示的行为进行定制，使其更容易符合原有应用的需求。

9.3 推广墙平台

游戏社交平台给游戏带来了新的特性，提升了游戏的可玩性，而手机广告平台能够让游戏嵌入广告，提高了游戏的收入。现在我们开始介绍第三个平台——推广墙平台。

推广墙平台是一个介于游戏社交平台和手机广告平台之间的平台。它是一个让玩家不需要购买游戏内虚拟货币，通过完成推广墙内的任务从而免费获得游戏内虚拟货币的平台。推广墙同时也是一种新的推广方式，通过这种推广方式，开发者可以获得额外的收入并增加游戏的曝光率。

WiOffer是微云提供的推广墙平台，目前仅包括Android版本。我们介绍的cocos2d-x引擎与推广墙平台的集成就是cocos2d-x引擎与WiOffer的集成。

WiOffer是微云推出的应用交叉推广与虚拟货币系统，类似于Tapjoy的Offer系统，开发者在自己的应用和游戏中嵌入WiOffer的SDK，展示微云推广墙，即可与其他应用之间相互推荐，让数千个软件帮助他推广应用，引起新用户增长的链式反应。开发者可以从中获得可观的推广收入，获得更多的用户，数十倍提升应用传播效果。

手机用户通过下载安装推广墙里面的应用和游戏，获得虚拟货币，用户可用其在应用内购买游戏关卡、道具、场景和皮肤等高级功能以及别的内容和服务，增强用户的活跃度与黏性，培养用户的应用内购买习惯，提高付费转化率，开发者结合使用WiGame的虚拟商店和支付系统，更能获得丰厚的虚拟物品销售收入。

9.4 技术准备

在前几章，我们已经介绍了cocos2d-x引擎是一款使用C++作为开发语言的游戏引擎。而微云的游戏社交平台、手机广告平台和推广墙平台在iOS上是使用Objective-C语言的，在Android上是

使用Java语言的。所以cocos2d-x引擎要与这几个平台集成，需要让cocos2d-x引擎能够直接调用Objective-C和Java的一些模块。现在我们分别介绍这两种调用方式。

9.4.1 cocos2d-x调用Objective-C

cocos2d-x引擎使用C++开发语言开发，那么cocos2d-x调用Objective-C也就是让C++调用Objective-C。

在之前的几章里，我们了解到基于cocos2d-x引擎开发的游戏如果想要把游戏发布到iOS上，就需要将代码放到Mac操作系统上进行编译，那么我们只要在写代码时，增加Mac做一个判断就可以调用Objective-C了。

我们直接以cocos2d-x引擎自带的一些代码为例分析C++如何调用Objective-C。下面是CCApplication_platform.h中的一段代码：

```
#include "CCPlatformConfig.h"

#if (CC_TARGET_PLATFORM == CC_PLATFORM_WIN32)
    #include "win32/CCApplication_win32.h"
#elif (CC_TARGET_PLATFORM == CC_PLATFORM_ANDROID)
    #include "android/CCApplication_android.h"
#elif (CC_TARGET_PLATFORM == CC_PLATFORM_IOS)
    # include "ios/CCApplication_ios.h"
#elif (CC_TARGET_PLATFORM == CC_PLATFORM_WOPHONE)
    #include "wophone/CCApplication_wophone.h"
#elif (CC_TARGET_PLATFORM == CC_PLATFORM_AIRPLAY)
   #include "airplay/CCApplication_airplay.h"
#else
    #error
#endif
```

在上述这段代码里，我们发现首先引用了头文件CCPlatformConfig.h，这个头文件是cocos2d-x引擎跨平台特性的基础，CCPlatformConfig.h里面定义了很多的常量，用于区分不同的操作系统。至于这些常量，可以参考第5章的具体描述。

然后根据跨平台的常量判断当前是什么操作系统：CC_TARGET_PLATFORM = CC_PLATFORM_WIN32表示当前系统是Windows系统，CC_TARGET_PLATFORM = CC_PLATFORM_ANDROID表示当前系统是谷歌的Android系统，CC_TARGET_PLATFORM = CC_PLATFORM_IOS表示当前系统是苹果的iOS系统，CC_TARGET_PLATFORM = CC_PLATFORM_WOPHONE表示当前系统是沃Phone系统，CC_TARGET_PLATFORM = CC_PLATFORM_AIRPLAY表示当前系统是AirPlay系统。

得到当前系统后，我们就可以在CC_PLATFORM_IOS这个分支内调用Objective-C的函数了。

在C++如何调用Objective-C的开发中，第一要紧的事情不是代码而是编译器选项，在做混合编译之前一定要把编译器的Compile Sources As选项改为Objective-C++。默认的选项是According to file type。如果选项是According to file type的话，一旦两种语言在一个游戏代码文件中相互调用，

就会报错。

当然，我们可以用C++代码调用方法，也可以用Objective-C调用方法。在这两种语言里对象都是指针，可以在任何地方使用。例如，C++类可以使用Objective-C对象的指针作为数据成员，Objective-C类也可以有C++对象指针作为实例变量。

正如你可以在Objective-C接口中声明C结构那样，你也可以在Objective-C接口中声明C++类。跟C结构一样，Objective-C接口中定义的C++类是全局范围的，不是Objective-C类的内嵌类（这与标准C提升嵌套结构定义为文件范围是一致的）。

为了允许基于语言条件化地编写代码，Objective-C编译器定义了`__cplusplus`和`__OBJC__`预处理器常量，分别指定C++和Objective-C。

C++与Objective-C互相调用时，关于文件名的原则是：Xcode的编译器允许一个源文件中同时包含C++和Objective-C源代码，但是这样的源文件的扩展名必须是“.mm”，否则编译器不能正确处理。这样源文件包含的代码统称为Objective-C++代码。一旦import的“.h”文件中也import了C++的“.h”文件，则对应的“.m”文件应该变成“.mm”文件。

9.4.2 cocos2d-x调用Java

cocos2d-x引擎是使用C++开发语言开发的，那么cocos2d-x调用Java也就是让C++调用Java。说到这里，大家就应该都知道了，这就是JNI的技术。

JNI（英文全称为Java Native Interface）的中文翻译为Java原生接口，它是Sun公司提供的Java与系统中的原生方法交互的技术（在Windows以及Linux系统中，实现Java与本地方法互调），目前只能由C/C++实现。从Java 1.1开始，JNI标准成为Java平台的一部分，它允许Java代码和其他语言写的代码进行交互。

JVM在屏蔽各种操作系统实际的差异性的同时，还提供了JNI技术，使得开发者可以通过Java程序（代码）调用操作系统相关的技术实现的库函数，从而与其他技术和系统交互，使用其他技术实现的系统的功能；同时其他技术和系统也可以通过JNI提供的相应原生接口调用Java应用系统内部实现的功能。

尽管使用Java与本地已编译的代码交互通常会丧失平台可移植性，但是，有些情况下这样做是可以接受的，甚至是必须的，比如，使用一些旧的库与硬件、操作系统进行交互，或者为了提高程序的性能。JNI标准至少保证本地代码能工作在任何Java虚拟机实现下。

在Windows系统上，一般可执行的应用程序都是基于Native的PE结构的，Windows上的JVM也是基于结构实现的。Java应用体系都是构建于JVM之上的，关系如图9-1所示。Linux平台Java体系与Windows平台上的类似。

对于应用本身来说，可以把JNI看做一个代理模式。对于开发者来说，需要使用C/C++实现一个代理程序（JNI程序）来实际操作目标原生函数，Java程序中则是JVM通过加载并调用此JNI程序来间接地调用目标原生函数。Java调用C++的关系如图9-2所示，而C++调用Java的关系如图

9-3所示。

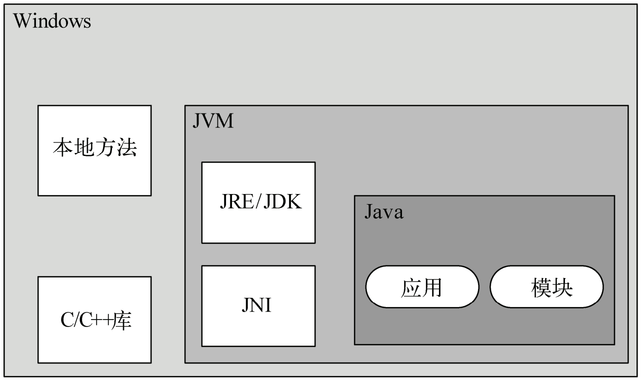


图9-1 Windows系统上的Java体系

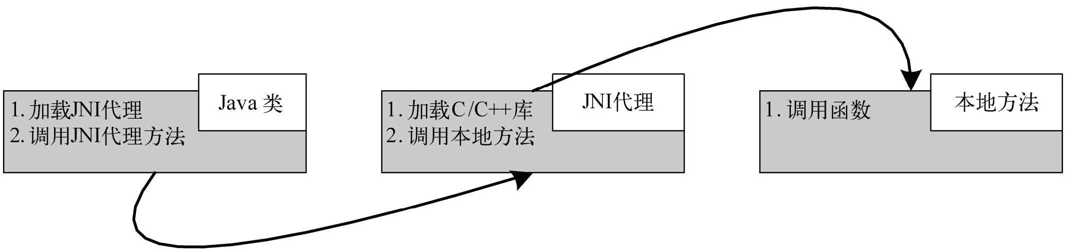


图9-2 Java调用C++关系

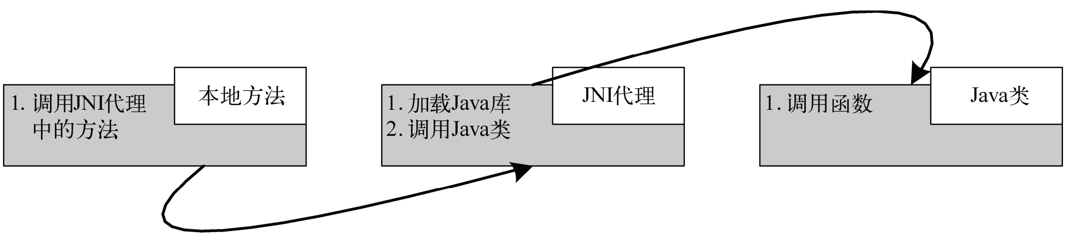


图9-3 C++调用Java关系

C++要调用Java程序必须先加载Java虚拟机，由Java虚拟机解释执行类文件。为了初始化Java虚拟机，JNI提供了一系列的接口函数，通过这些函数可以很方便地将虚拟机加载到内存中。

(1) **初始化虚拟机**

函数为`jint JNI_CreateJavaVM(JavaVM **pvm, void **env, void args)`。

第一个参数`JavaVM **pvm`是Java虚拟机的指针，第二个参数`void **env`是贯穿整个调用过程的一个参数，因为后面的所有函数都需要这个参数，第三个参数`void args`在JDK1.1和JDK1.2之后的版本上有些不同。在JDK 1.1中总是指向一个结构`JDK1_1InitArgs`，而这个结构无法在所有版本的虚拟机中进行无缝移植。所以为了保证JNI代码的可移植性，建议使用JDK 1.2

的方法来初始化虚拟机。表9-2就是在JDK1.1和JDK1.2下初始化虚拟机的代码。

表9-2 在JDK1.1和JDK1.2下初始化虚拟机的代码

JDK的版本	初始化虚拟机

JDK 1.1

```
int main()
{
    JNIEnv *env;
    JavaVM *jvm;
    JDK1_1InitArgs vm_args;
    jint res;

    //版本号
    vm_args.version = 0x00010001;

    //获取缺省的虚拟机初始化参数
    JNI_GetDefaultJavaVMInitArgs(&vm_args);

    //创建虚拟机
    res = JNI_CreateJavaVM(&jvm,&env,&vm_args);
    if (res < 0) {
        exit(1);
    }

    ///
    (*jvm)->DestroyJavaVM(jvm);
}
```

JDK 1.2

```
int main()
{
    int res;
    JavaVM *jvm;
    JNIEnv *env;
    JavaVMInitArgs vm_args;

    JavaVMOption options[3];
    //设置初始化参数
    options[0].optionString = "-Djava.compiler=NONE";
    options[1].optionString = "-Djava.class.path=.";
    options[2].optionString = "-verbose:jni";

    //版本号
    vm_args.version = JNI_VERSION_1_2;
    vm_args.nOptions = 3;
    vm_args.options = options;
    vm_args.ignoreUnrecognized = JNI_TRUE;

    res = JNI_CreateJavaVM(&jvm, (void**)&env, &vm_args);
    if (res < 0)
    {
        exit(1);
    }
    //释放虚拟机资源
    (*jvm)->DestroyJavaVM(jvm);
}
```

(2) 获取指定对象的类定义

若知道类名则通过类名得到类定义，使用jclass FindClass(JNIEnv *env, const char *name);来获取。

若不知道类名则通过对象直接得到类定义，使用jclass GetObjectClass(JNIEnv *env, jobject obj)来获取。

(3) 获取要调用的方法

获得非静态方法使用jmethodID GetMethodID(JNIEnv *env, jclass clazz, const char *name, const char *sig)。

获得静态方法使用jmethodID GetStaticMethodID(JNIEnv *env, jclass clazz, const char *name, const char *sig)。

第一个参数JNIEnv *env是初始化虚拟机获取的JNI环境的指针，第二个参数jclass clazz是已获取到的类定义，第三个参数const char *name是方法名，第四个参数const char *sig是方法的定义，我们知道Java支持多态，同名方法通过第四个参数来定位得到具体的方法。

(4) 调用Java类方法

调用非静态函数使用Call<type>Method(JNIEnv *env, jobject obj, jmethodID methodID, ...)、Call<type>MethodA(JNIEnv *env, jobject obj, jmethodID methodID, jvalue *args);、Call<type>MethodV(JNIEnv *env, jobject obj, jmethodID methodID, va_list args)方法。这里大家可能会看不懂<type>，其实它只是一个通配符，比如Call<type>Method，表示在JNI里面有下面这些函数：CallStaticVoidMethod、CallStaticObjectMethod、CallStaticBooleanMethod、CallStaticByteMethod、CallStaticCharMethod、CallStaticShortMethod、CallStaticIntMethod、CallStaticLongMethod、CallStaticFloatMethod、CallStaticDoubleMethod。

调用静态函数使用CallStatic<type>Method(JNIEnv *env, jobject obj, jmethodID methodID, ...)、CallStatic<type>MethodA(JNIEnv *env, jobject obj, jmethodID methodID, jvalue *args); 、CallStatic<type>MethodV(JNIEnv *env, jobject obj, jmethodID methodID, va_list args)方法。关于通配符<type>解释同上。

(5) 获得类属性的定义

获取静态属性使用jfieldID GetStaticFieldID(JNIEnv *env, jclass clazz, const char *name, const char *sig)、NativeType GetStatic<type>Field(JNIEnv *env, jclass clazz,jfieldID fieldID)方法。关于通配符<type>解释同上。

获取非静态属性使用jfieldID GetFieldID(JNIEnv *env, jclass clazz, const char *name, const char *sig)、NativeType Get<type>Field(JNIEnv *env, jobject obj, jfieldID fieldID)方法。关于通配符<type>解释同上。

JNI除了提供上述这些基本的操作外，还提供了数组处理、异常处理和多线程调用等处理方

案。这样就可以在C++中使用Java开发的方法了。

9.5 案例实现

至此我们已经掌握了cocos2d-x引擎调用Objective-C和Java的基本方法了。现在我们就开始用具体的游戏场景完成cocos2d-x引擎与游戏社交平台、手机广告平台和推广墙平台的集成。

9.5.1 场景分析

在某一个游戏中，玩家需要分享自己的游戏成果，需要向其他的游戏玩家发起挑战，需要与其他玩该游戏的玩家进行交流，也需要向游戏开发者提出意见。同样，在这个游戏中，开发者需要得到玩家的信息，需要得到玩家的反馈意见，需要嵌入一些广告进行盈利，也需要推广自己的游戏产品。

基于上述两种情况，开发者非常需要一套能满足社交、广告和推广的解决方案。开发者对此做出的需求分析结果如下。

(1) 游戏内增加社交的入口，让玩家可以关注好友的最新情况、分享应用以及向别的玩家挑战。

(2) 游戏中在不影响用户体验的前提下增加广告展示的区域，用于获取收入。

(3) 游戏增加推广墙功能，可以与其他游戏或应用相互推荐，让数千个游戏或应用帮助推广游戏，引起新用户增长的链式反应，从中获得良好的推广收入，获得更多的用户，数十倍提升应用传播效果。

9.5.2 环境准备

对本场景需要的开发环境的准备，这里的环境包括以下4个。

(1) cocos2d-x引擎的开发环境，此部分可参考第4章的内容。

(2) 游戏社交平台（WiGame）的SDK，包括用于iOS平台的WiGame_sdk_iOS_3.0_release.zip文件和用于Android的WiGame_sdk_android_3.0.4.zip。

(3) 手机广告平台（WiAd）的SDK，包括用于iOS平台的WiAd_sdk_iOS_2.1.0_release.zip文件和用于Android的WiAd_sdk_android_1.2.3.zip。

(4) 推广墙平台（WiOffer）的SDK，包括用于Android的wioffer_android_sdk_1.0.4.zip。

关于上面提到的微云各个平台的SDK，可访问微云官方网站www.wiyun.com搜索，或者直接访问http://www.wiyun.com/a/SDK/index.html下载到最新版本的SDK。

9.5.3 游戏设计

根据前面的场景描述，我们现在对游戏进行设计。首先在游戏的首页至少包括两个区域，一是广告区域，广告区域用于展示手机广告平台（WiAd）的广告；二是菜单区域，菜单区域至少

需要包括“游戏社区”和“推广墙”两个按钮，也就是如图9-4所示的界面流转图。

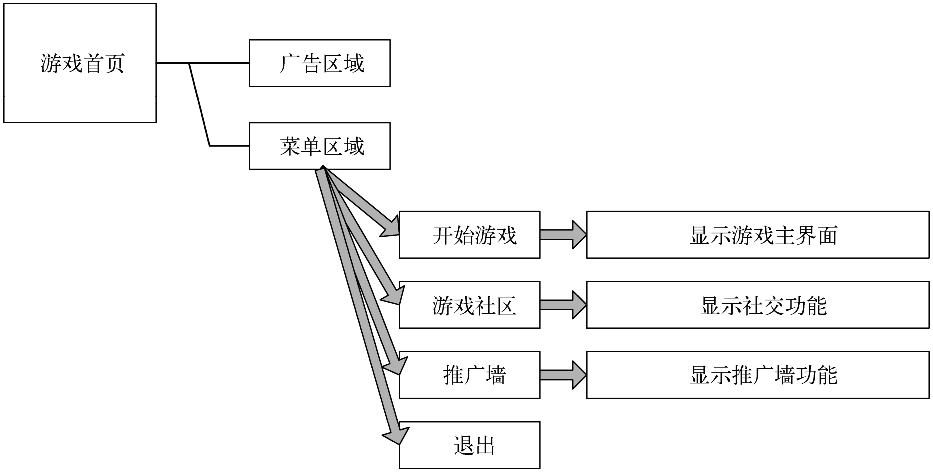


图9-4　游戏界面流转

9.5.4　游戏实现

本章的核心内容是cocos2d-x引擎与游戏社交平台、手机广告平台和推广墙平台的集成，我们就不花精力在游戏本身的制作上了。因此我们选择第4章提到的那个Hello World游戏，对其稍作调整。

第一步，去除FPS显示，修改AppDelegate.cpp文件里的下列代码，将`setDisplayFPS(true)`改成`setDisplayFPS(false)`。

```
//true：显示FPS，false：不显示FPS
pDirector->setDisplayFPS(false);
```

第二步，调整一下游戏的首页，让其更好看一点。首页的`init`函数代码修改为：

```
bool HelloWorld::init()
{
    //调用父类的init方法
    if ( !CCLayer::init() )
    {
        return false;
    }

    //增加一个标签，文字为：社交游戏
    CCLabelTTF *pLabel = CCLabelTTF::labelWithString("社交游戏", "Thonburi", 34);

    //获取屏幕大小
    CCSize size = CCDirector::sharedDirector()->getWinSize();

    //设置Label的位置
```

```
    pLabel->setPosition( ccp(size.width / 2, size.height - 20) );

    //将Label增加到当前层中
    this->addChild(pLabel, 1);

    //设置背景图片为HelloWorld.png
    CCSprite *pSprite = CCSprite::spriteWithFile("HelloWorld.png");

    //设置背景图片的位置
    pSprite->setPosition( ccp(size.width/2, size.height/2) );

    //将背景图片增加到当前层中
    this->addChild(pSprite, 0);

    return true;
}
```

根据设计，在游戏首页需要增加下述按钮，在Windows平台上菜单按钮为“开始游戏”和“退出”，在iOS平台上菜单按钮为“开始游戏”、“游戏社区”和“退出”，在Android平台上菜单按钮为“开始游戏”、“游戏社区”、“推广墙”和“退出”，在沃Phone平台上菜单按钮为“开始游戏”和“退出”，如表9-3所示。

表9-3 菜单按钮

平 台	菜单按钮
Windows平台	“开始游戏”和“退出”
iOS平台	“开始游戏”、“游戏社区”和“退出”
Android平台	“开始游戏”、“游戏社区”、“推广墙”和“退出”
沃Phone平台	“开始游戏”和“退出”

我们在Hello World游戏的HelloWorldScene.cpp的`init`函数里面增加下面的代码：

```
//当前平台是Windows平台或者沃Phone平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_WIN32 || CC_TARGET_PLATFORM == CC_PLATFORM_
    WOPHONE)
    CCMenuItemFont::setFontSize(24);
    CCMenuItemFont *start = CCMenuItemFont::itemFromString("开始游戏", this, menu_selector
    (HelloWorld::menuCallbackStart));
    CCMenuItemFont *exit = CCMenuItemFont::itemFromString("退出", this, menu_selector
    (HelloWorld::menuCallbackExit));

    start->setAnchorPoint(CCPointZero);
    start->setPosition(ccp(5, 80));

    exit->setAnchorPoint(CCPointZero);
    exit->setPosition(ccp(5, 50));

    CCMenu *menu = CCMenu::menuWithItems( start, exit, NULL);
    menu->setPosition( CCPointZero );
```

```
    this->addChild(menu, 2);
#endif

//当前平台是iOS平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_IOS)
    CCMenuItemFont::setFontSize(24);
    CCMenuItemFont *start = CCMenuItemFont::itemFromString("开始游戏", this, menu_ selector
    (HelloWorld::menuCallbackStart));
    CCMenuItemFont *sns = CCMenuItemFont::itemFromString("游戏社区", this, menu_ selector
    (HelloWorld::menuCallbackSNS));
    CCMenuItemFont *exit = CCMenuItemFont::itemFromString("退出", this, menu_ selector
    (HelloWorld::menuCallbackExit));

    start->setAnchorPoint(CCPointZero);
    start->setPosition(ccp(5, 110));

    sns->setAnchorPoint(CCPointZero);
    sns->setPosition(ccp(5, 80));

    exit->setAnchorPoint(CCPointZero);
    exit->setPosition(ccp(5, 50));

    CCMenu *menu = CCMenu::menuWithItems( start,  sns, exit, NULL);
    menu->setPosition( CCPointZero );
    this->addChild(menu, 2);
#endif

//当前平台是Android平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_ANDROID)
    CCMenuItemFont::setFontSize(24);
    CCMenuItemFont *start = CCMenuItemFont::itemFromString("开始游戏", this, menu_selector
    (HelloWorld::menuCallbackStart));
    CCMenuItemFont *sns = CCMenuItemFont::itemFromString("游戏社区", this, menu_selector
    (HelloWorld::menuCallbackSNS));
    CCMenuItemFont *offer = CCMenuItemFont::itemFromString("推广墙", this, menu_selector
    (HelloWorld::menuCallbackOffer));
    CCMenuItemFont *exit = CCMenuItemFont::itemFromString("退出", this, menu_selector
    (HelloWorld::menuCallbackExit));

    start->setAnchorPoint(CCPointZero);
    start->setPosition(ccp(5, 140));

    sns->setAnchorPoint(CCPointZero);
    sns->setPosition(ccp(5, 110));

    offer->setAnchorPoint(CCPointZero);
    offer->setPosition(ccp(5, 80));

    exit->setAnchorPoint(CCPointZero);
    exit->setPosition(ccp(5, 50));

    CCMenu *menu = CCMenu::menuWithItems( start,  sns, offer, exit, NULL);
    menu->setPosition( CCPointZero );
    this->addChild(menu, 2);
#endif
```

每一个菜单需要设置被点击后的回调函数，也就是上面代码里面的HelloWorld:: menuCallbackStart、HelloWorld::menuCallbackSNS、menuCallbackOffer、HelloWorld::menuCallbackExit，这4个函数需要在HelloWorldScene.h里面申明，申明的代码如下：

```
//开始游戏的回调函数
void menuCallbackStart(CCObject *sender);
//游戏社交的回调函数
void menuCallbackSNS(CCObject *sender);
//推广墙的回调函数
void menuCallbackOffer(CCObject *sender);
//退出的回调函数
void menuCallbackExit(CCObject *sender);
```

点击“开始游戏”按钮，游戏进入主界面。假如游戏主界面的场景是GameScene，那么按钮的实现代码如下：

```
//开始游戏的回调函数
void HelloWorld::menuCallbackStart(CCObject *sender)
{
    CCScene *pScene = GameScene::scene();
    if (pScene)
    {
        CCDirector::sharedDirector()->pushScene( pScene);
    }
}
```

点击“退出”按钮，游戏关闭。按钮的实现代码如下：

```
//退出的回调函数
void HelloWorld::menuCallbackExit(CCObject *sender)
{
    CCDirector::sharedDirector()->end();

    #if (CC_TARGET_PLATFORM == CC_PLATFORM_IOS)
        exit(0);
    #endif
}
```

点击“游戏社区”按钮，将调用游戏社交平台的相关功能，具体实现请阅读本节下面的“游戏社交平台集成”部分。

点击“推广墙”按钮，将调用推广墙平台的相关功能，具体实现请阅读本节下面的“推广墙平台集成”部分。

同样，根据设计，需要在游戏首页嵌入广告区域，具体实现请阅读本节下面的“手机广告平台集成”部分。

1. 游戏社交平台集成

在9.1节里，我们已经对游戏社交平台WiGame做了比较详细的描述。现在我们就把这个平台

集成到我们的游戏中。

查看WiGame在iOS和Android平台上的SDK，我们发现两者虽然实现的语言不同，但是在整体设计上还是大同小异的。首先通过一个初始化函数设置WiGame的一些全局参数。在iOS平台上是WiGame的initWithAppKey函数，在Android平台上是WiGame的init函数。然后就可以通过各自的语言调用不同的接口来调用WiGame的功能了。

由于初始化函数与平台的耦合度比较高，所以这块代码不做封装。表9-4就是iOS平台和Android平台上WiGame的初始化代码。

表9-4　iOS平台和Android平台上WiGame的初始化代码

平　　台	触 发 点	初始化代码
iOS平台	applicationDid-FinishLaunching方法	//初始化WiGame平台 [[WiGame sharedInstance] initWithAppKey:@"<填入应用的app key>" secretKey:@"<填入应用的secret key>" appVersion:@"<填入应用版本,格式是x.y >" testMode:NO];

Android平台	onCreate方法	//初始化WiGame平台
		WiGame.init(this, "填入应用的app key", "填入应用的secret key", "填入应用版本,格式是x.y", false, false);

接下来，我们就封装WiGame的“启动游戏社交平台”功能。在iOS平台上是WiGame的launchPanel函数，在Android平台上是WiGame的startUI函数。

首先定义一个公共的头文件CommonWiGame.h，以后调用“启动游戏社交平台”功能时，只需要引用这个公共的头文件即可，CommonWiGame.h代码如下：

```
#ifndef __CommonWiGame_h__
#define __CommonWiGame_h__

#include "cocos2d.h"

#ifdef __cplusplus
extern "C" {
#endif

//定义启动游戏社交平台函数
void startUI();

#ifdef __cplusplus
}
#endif

#endif // __CommonWiGame_h__
```

与CommonWiGame.h配套的CommonWiGame.cpp的实现如下， CommonWiGame.cpp提供了iOS和Android两个平台的实现。

```
#include "CommonWiGame.h"
//当前平台是iOS平台
```

```
#if (CC_TARGET_PLATFORM == CC_PLATFORM_IOS)
    #include "WiGame.h"
#endif
//当前平台是Android平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_ANDROID)
    #include "cocos2d.h"
#endif

#ifdef __cplusplus
extern "C" {
#endif

//当前平台是iOS平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_IOS)
    //启动游戏社交平台
    void startUI()
    {
        [[WiGame sharedInstance] launchPanel];
    }
#endif

//当前平台是Android平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_ANDROID)
    //启动游戏社交平台
    void startUI()
    {
        //获取JNI环境
        JNIEnv *env = NULL;
            if (gJavaVM->GetEnv((void**) &env, JNI_VERSION_1_4) != JNI_OK)
            {
                int status = gJavaVM->AttachCurrentThread(&env, NULL);
                if(status < 0)
                {
                    return;
                }
            }else
            {
                return;
            }

        //寻找类和函数
        jclass clazz = env->FindClass("com/wiyun/game/WiGame");
        jmethodID mid = env->GetStaticMethodID(clazz, "startUI", "()V");

        //调用本地方法
        env->CallStaticVoidMethod(clazz, mid);

        //删除不再需要的引用
        env->DeleteLocalRef(clazz);
    }
#endif

#ifdef __cplusplus
}
#endif
```

完成上述封装后，我们就可以在各自的平台上针对集成后的代码进行编译了。

● 在iOS平台编译

第一步，在项目的链接库（Linked Libraries）上增加引用库：MediaPlayer.framework、SystemConfiguration.framework、MapKit.framework、CoreLocation.framework、CFNetwork.framework、QuartzCore.framework、StoreKit.framework、libsqlite3.dylib。

第二步，将WiGame提供给iOS平台的SDK（WiGame_sdk_iOS_3.0_release.zip）解压缩得到SDK的文件，然后将sdk目录拖放到工程下，在弹出的对话框中选中“Recursively create groups for any added folders”。

第三步，在项目属性（Project Info）中找到“Other Linker Flags”选项，在其中添加选项-ObjC和all_load。

● 在Android平台编译

第一步，将WiGame提供给Android平台的SDK（WiGame_sdk_android_3.0.4.zip）解压缩得到SDK的文件，然后将res目录下的所有文件复制到游戏Android项目的res目录内。

第二步，将libs目录下的WiGame.jar复制到游戏Android项目内，并设置依赖关系。

第三步，修改AndroidManifest.xml，增加游戏社交平台展示时需要的活动Activity，增加的活动Activity内容包括：

```
com.wiyun.game.AccountRetrieval
com.wiyun.game.BidPicker
com.wiyun.game.ChangeMyPortrait
com.wiyun.game.ComposeTopic
com.wiyun.game.CropImage
com.wiyun.game.DLCDownloader
com.wiyun.game.DownloadBlob
com.wiyun.game.FullImageGallery
com.wiyun.game.SaveGameDialog
com.wiyun.game.Home
com.wiyun.game.LoadGameDialog
com.wiyun.game.Login
com.wiyun.game.MyBagDialog
com.wiyun.game.OAuth
com.wiyun.game.PurchaseDialog
com.wiyun.game.SendChallenge
com.wiyun.game.SubmitScore
com.wiyun.game.SwitchAccount
com.wiyun.game.UseAnotherAccount
com.wiyun.game.UserMap
```

第四步，修改AndroidManifest.xml，增加游戏社交平台运行时需要的权限，增加的权限内容如下：

```
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

2. 手机广告平台集成

在9.2节里，我们已经对手机广告平台WiAd做了比较详细的描述。现在我们就把这个平台集成到我们的游戏中。

查看WiAd在iOS和Android平台上的SDK，我们发现两者虽然实现的语言不同，但是在整体设计上还是大同小异的。广告最终都是需要使用某一个区域进行展示的，iOS平台用到了iOS平台的View，Android平台用到了Android平台的View。所以我们需要做一个封装，用一个公共的方法通过各自的语言调用不同的接口来调用广告的展示功能。

接下来，我们就封装WiAd的“广告展示”功能，在iOS平台上是WiAdView的`requestAd`函数，在Android平台上是WiGame的`requestAd`函数。

在前面定义的公共头文件CommonWiGame.h内增加下列内容，以后调用“请求广告”功能时，只需要引用这个公共的头文件。

```
#ifndef __CommonWiGame_h__
#define __CommonWiGame_h__

#include "cocos2d.h"

#ifdef __cplusplus
extern "C" {
#endif

//定义启动游戏社交平台函数
void startUI();
#if (CC_TARGET_PLATFORM == CC_PLATFORM_ANDROID)
    //定义打开Offer列表函数
    void showOffers();
#endif
//定义请求广告函数
void requestAd();

#ifdef __cplusplus
}
#endif

#endif // __CommonWiGame_h__
```

与CommonWiGame.h配套的CommonWiGame.cpp的实现如下，CommonWiGame.cpp提供了iOS和Android两个平台的实现。

```
#include "CommonWiGame.h"
```

```
//当前平台是iOS平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_IOS)
    #include "WiGame.h"
#endif
//当前平台是Android平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_ANDROID)
    #include "cocos2d.h"
#endif

#ifdef __cplusplus
extern "C" {
#endif

//当前平台是iOS平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_IOS)
    //启动游戏社交平台
    void startUI()
    {
        [[WiGame sharedInstance] launchPanel];
}

//请求广告
    void requestAd()
    {
        WiAdView *adView = [WiAdView adViewWithResId:@"广告位ID" style:kWiAdViewStyle-
        Banner320_50];
         [self.view addSubview:adView];
          [adView requestAd];
}

#endif

//当前平台是Android平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_ANDROID)
    //启动游戏社交平台
    void startUI()
    {
        //获取JNI环境
        JNIEnv *env = NULL;
            if (gJavaVM->GetEnv((void**) &env, JNI_VERSION_1_4) != JNI_OK)
            {
                int status = gJavaVM->AttachCurrentThread(&env, NULL);
                if(status < 0)
                {
                    return;
                }
            }else
            {
                return;
            }

        //寻找类和函数
```

```
    jclass clazz = env->FindClass("com/wiyun/game/WiGame");
    jmethodID mid = env->GetStaticMethodID(clazz, "startUI", "()V");

    //调用本地方法
    env->CallStaticVoidMethod(clazz, mid);

    //删除不再需要的引用
    env->DeleteLocalRef(clazz);
}

//打开Offer列表
void showOffers()
{
    //获取JNI环境
    JNIEnv *env = NULL;
    if (gJavaVM->GetEnv((void**) &env, JNI_VERSION_1_4) != JNI_OK)
    {
            int status = gJavaVM->AttachCurrentThread(&env, NULL);
            if(status < 0)
            {
                return;
            }
        }else
        {
            return;
        }

    //寻找类和函数
    jclass clazz = env->FindClass("com/wiyun/offer/WiOffer");
    jmethodID mid = env->GetStaticMethodID(clazz, "showOffers", "()V");

    //调用本地方法
    env->CallStaticVoidMethod(clazz, mid);

    //删除不再需要的引用
    env->DeleteLocalRef(clazz);
}

//请求广告
void requestAd ()
{
    //获取JNI环境
    JNIEnv *env = NULL;
    if (gJavaVM->GetEnv((void**) &env, JNI_VERSION_1_4) != JNI_OK)
    {
            int status = gJavaVM->AttachCurrentThread(&env, NULL);
            if(status < 0)
            {
                return;
            }
        }else
        {
```

```
            return;
        }

    //寻找类和函数
    jclass clazz = env->FindClass("com/wiyun/game/WiGame");
    jmethodID mid = env->GetStaticMethodID(clazz, "requestAd", "()V");

    //调用本地方法
    env->CallStaticVoidMethod(clazz, mid);

    //删除不再需要的引用
    env->DeleteLocalRef(clazz);
}

#endif

#ifdef __cplusplus
}
#endif
```

完成上述封装后，我们就可以在各自的平台上针对集成后的代码进行编译了。

- 在iOS平台编译

第一步，在项目的链接库（Linked Libraries）上增加引用库：MediaPlayer.framework和SystemConfiguration.framework。

第二步，将WiAd提供给iOS平台的SDK（WiAd_sdk_iOS_2.1.0_release.zip）解压缩得到SDK的文件，然后将sdk目录拖放到工程下，在弹出的对话框中选中"Recursively create groups for any added folders"。

第三步，在项目属性（Project Info）中找到"Other Linker Flags"选项，在其中添加选项-ObjC和all_load。

- 在Android平台编译

第一步，将WiAd提供给Android平台的SDK（WiAd_sdk_android_1.2.3.zip）解压缩得到SDK的文件，然后将res目录下的所有文件复制到游戏Android项目的res目录内。

第二步，将libs目录下的WiAd.jar复制到游戏Android项目内，并设置依赖关系。

第三步，修改AndroidManifest.xml，增加手机广告平台运行时需要的权限，增加的权限内容如下：

```
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

3. 推广墙平台集成

在9.3节里，我们已经对推广墙平台WiOffer做了比较详细的描述。现在我们就把这个平台集

成到我们的游戏中。

微云WiOffer暂时只支持了Android平台。同样，首先通过一个初始化函数设置WiOffer的一些全局参数。在Android平台上是WiOffer的init函数。然后就可以通过各自的语言调用不同的接口来调用WiOffer的功能了。

由于初始化函数与平台的耦合度比较高，所以这块代码不做封装。表9-5就是Android平台上WiOffer的初始化代码。

表9-5　Android平台上初始化WiOffer的代码

平　　台	触 发 点	初始化代码

Android平台	onCreate方法	//初始化WiOffer平台
		WiOffer.init(this, "填入应用的app key", "填入应用的secret key");

接下来，我们就封装WiOffer的“打开Offer列表”功能。在Android平台上是WiOffer的showOffers函数。

在前面定义的公共头文件CommonWiGame.h内增加下列内容，以后调用“打开Offer列表”功能时，只需要引用这个公共的头文件。

```
#ifndef __CommonWiGame_h__
#define __CommonWiGame_h__

#include "cocos2d.h"

#ifdef __cplusplus
extern "C" {
#endif

#if (CC_TARGET_PLATFORM == CC_PLATFORM_ANDROID)
    //定义打开Offer列表函数
    void showOffers();
#endif

#ifdef __cplusplus
}
#endif

#endif // __CommonWiGame_h__
```

与CommonWiGame.h配套的CommonWiGame.cpp也需要实现showOffers函数。由于不需要支持iOS，所以代码会相对简单一点。

```
#include "CommonWiGame.h"
//当前平台是iOS平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_IOS)
    #include "WiGame.h"
#endif
```

```
//当前平台是Android平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_ANDROID)
    #include "cocos2d.h"
#endif

#ifdef __cplusplus
extern "C" {
#endif

//当前平台是Android平台
#if (CC_TARGET_PLATFORM == CC_PLATFORM_ANDROID)

    //打开Offer列表
    void showOffers ()
    {
        //获取JNI环境
        JNIEnv *env = NULL;
            if (gJavaVM->GetEnv((void**) &env, JNI_VERSION_1_4) != JNI_OK)
            {
                int status = gJavaVM->AttachCurrentThread(&env, NULL);
                if(status < 0)
                {
                    return;
                }
            }else
            {
                return;
            }

        //寻找类和函数
        jclass clazz = env->FindClass("com/wiyun/offer/WiOffer");
        jmethodID mid = env->GetStaticMethodID(clazz, "showOffers", "()V");

        //调用本地方法
        env->CallStaticVoidMethod(clazz, mid);
        //删除不再需要的引用
        env->DeleteLocalRef(clazz);
    }
#endif

#ifdef __cplusplus
}
#endif
```

完成上述封装后，我们就可以在Android平台上针对集成后的代码进行编译了。

- 在Android平台编译

第一步，将WiOffer提供给Android平台的SDK（wioffer_android_sdk_1.0.1.zip）解压缩得到SDK的文件，然后将libs目录下的WiOffer.jar复制到游戏Android项目内，并设置依赖关系。

第二步，修改AndroidManifest.xml，增加推广墙平台展示时需要的活动Activity，增加的活动

Activity内容包括：com.wiyun.offer.OfferList。

第三步，修改AndroidManifest.xml，增加推广墙平台运行时需要的权限，增加的权限内容如下：

```
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="anVBdroid.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

9.5.5 场景总结

通过这个案例，我们已经成功地将游戏社交平台、手机广告平台和推广墙平台提供的功能集成到游戏内。那么接下来就可以将自行发布运营，或者授权给微云合作发布、运营及推广，轻松获取更多的用户和更可观的收入。

9.6 总结

本章首先介绍游戏社交平台、手机广告平台和推广墙平台，然后学习了如何让cocos2d-x引擎调用Objective-C和Java提供的功能，最后以一个简单的案例实现了cocos2d-x引擎调用这3个平台的功能。

从下一章开始，我们将会以完整的篇幅介绍游戏制作过程，大家可以看到游戏是如何一步一步设计并开发出来的。

第10章

“魔塔”案例之基础篇

通过前面各章的学习，我们已经了解了cocos2d-x引擎的基础知识、高级特性、周边工具和交叉编译等知识。掌握了这些知识，我们就有了开发手机游戏的基础。现在本章将带领大家学习一个完整的案例——魔塔。我们会讲述一个完整的游戏制作过程，将各个步骤写得足够细致，即使你没有任何游戏开发经验，也可以读懂。

我们选择的游戏是魔塔，魔塔是一款非常经典和耐玩的益智类RPG游戏。笔者在上大学时就疯狂地迷恋上这一类型的游戏了。从Flash版本的24层魔塔，到新新魔塔，到广大魔塔制作者用RMXP/RMVP开发出的魔塔，风格各有不同，玩起来也都有不一样的感觉。直到有一天，我们决定将魔塔游戏移植到手机平台上。这种不需要复杂操作，可以完美利用碎片时间的游戏将十分符合手机用户的使用习惯。我们希望通过这一章抛砖引玉，帮助开发者方便地开发跨平台的此类游戏。

10.1 先熟悉一下游戏

如果你从来没有接触过魔塔游戏，可以上网搜索“魔塔”关键字，会有很多Flash版本的魔塔，请先体验一下。

现在，相信你已经对魔塔游戏有所了解，下面我们来概括一下。

(1) 这是一款固定数值计算的益智RPG游戏。

(2) 战斗结果是定死的，玩家需要计算损耗的血量与实际的收益（战斗后得到的宝物、金钱及经验）相比是否划算。

(3) 游戏里有各种RPG元素，包括地图、玩家扮演的角色、人物属性（攻击、防御、生命、金钱和经验等）、怪物、消耗性物品（血瓶和钥匙）、装备（剑和盾）以及剧情（对话和任务）等。

在下面我们制作的游戏中就将会包含这些元素。好了，让我们开始吧。

10.2 准备工作

我们用到的工具主要包括以下几种。

(1) 一台安装了Windows系统的电脑，如果需要编译iOS的版本，那么还需要准备一台安装了Mac系统的电脑。

(2) cocos2d-x的开发环境，关于开发环境的安装请看之前章节的内容。

(3) TMX地图编辑器，我们使用的是Tiled的QT版本，软件的官方网站是http://www.mapeditor.org/。

(4) 图片编辑工具PhotoShop。

10.3　绘制最简单的游戏地图

首先，我们打开cocos2d-x工程，使用"Cocos2d-win32 Application"模板新建一个基于cocos2d-x引擎的win32项目，命名为MTGame，如图10-1所示。

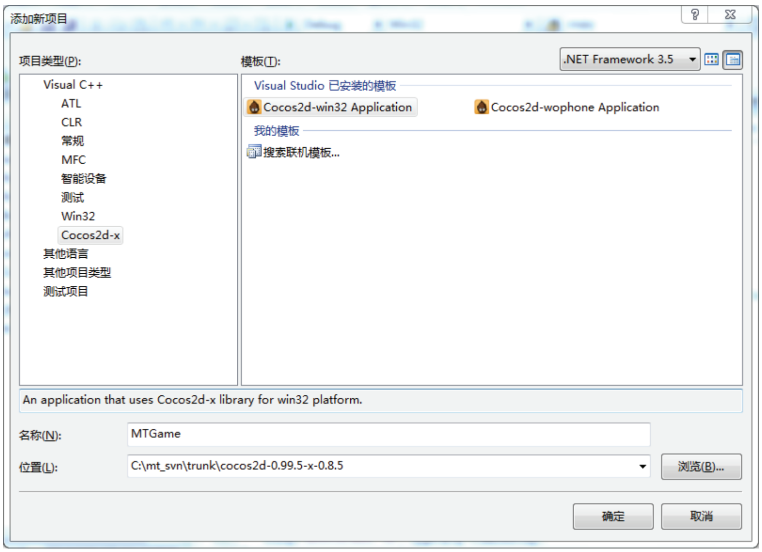


图10-1　新建项目

这款游戏不需要物理引擎Box2D和Chipmunk的支持，可以在新建项目的向导中将它们取消。CocosDenshion声音模块需要选中。新建完工程，我们可以先运行一下，保证一切正常。顺利的话，编译后可以直接看到cocos2d-x的logo，以及一个用来关闭程序的"×"。现在我们把这个新建的工程放在一旁，开始绘制游戏地图。

首先打开Tiled编辑器（假设你已经下载了，若没有下载请去前一节找下载链接）。新建一张地图。地图方向选择"正常"，代表垂直90°视角。地图大小为13×13，表示地图的宽和高皆为13个图块。块大小为32×32代表每个图块的尺寸，单位是像素，如图10-2所示。

新建完地图后，需要准备一张图片素材用来绘制地面和墙壁。假设我们已经准备好了，素材文件为terrain.png，其中包括了两个图块，第一个是地面（灰色岩石地面），第二个是墙壁（棕色

砖墙），宽和高都是32像素，如图10-3所示（左边是灰色岩石地面，右边是棕色砖墙）。



图10-2 新建地图

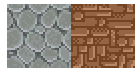


图10-3 素材terrain.png

将素材文件terrain.png复制到MTGame工程下面的Resource目录里，然后通过菜单栏的“地图→新图块”添加图片素材，如图10-4所示。



图10-4 导入图块

导入完成后，可以看到Tiled编辑器右侧下方的图块面板中出现了terrain标签。现在我们就可以使用导入的图块进行绘制了。选择工具栏中的填充工具（如图10-5所示），再选中terrain里的第一个图块，点击地图上任意位置，地面层就被我们填充成为灰色的岩石地面了。

接下来开始绘制墙壁。先在右侧图层面板的“块层1”上双击重命名为floor（注意，这个名字需要记住，并且取英文，在cocos2d-x引擎里解析时需要用到）。然后再新建一个块层，重命名为wall。wall层在floor层的上方，将来引擎绘制的时候，上方的层也会覆盖下面的图层。接着选择图章刷工具（如图10-6所示），选中棕色砖墙图块，开始绘制墙壁。



图10-5 填充工具



图10-6 图章刷工具

好了，现在是自由发挥时间。编辑完地图之后别忘了将地图保存到MTGame工程下面的Resource目录下面，先命名为0.tmx。我们完成的其中一个地图如图10-7所示。

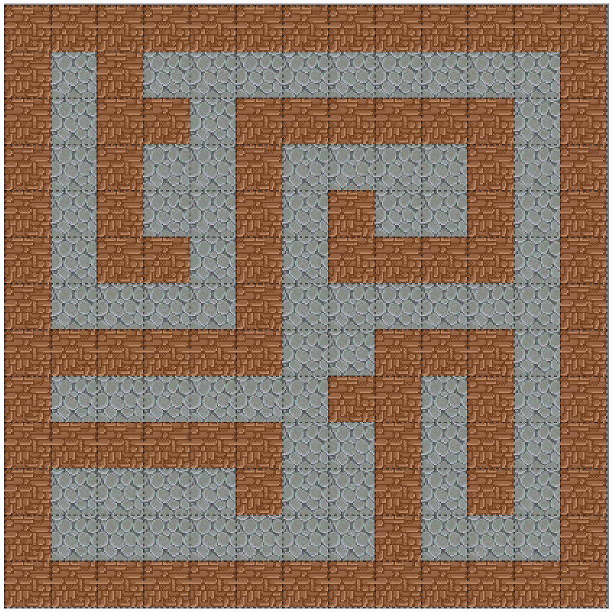


图10-7　完成后的地图

现在，我们把编辑好的游戏地图添加到游戏中。打开MTGame中的HelloWorldScene.cpp，将方法`init`中多余的代码删除，仅保留"退出"按钮。然后使用cocos2d-x引擎的`CCTMXTiledMap`类解析地图。

```
//解析tmx地图
CCTMXTiledMap *map = CCTMXTiledMap::tiledMapWithTMXFile("0.tmx");
addChild(map);
```

好了，仅仅需要上面的两行代码，地图就可以显示在游戏中了，运行效果如图10-8所示。

最后再提几个在绘制游戏地图时应注意的问题。

(1) Tiled编辑器支持在同一个块层使用多个图块纹理，但cocos2d-x引擎的`CCTMXTiledMap`类仅支持一个块层对应一个纹理。所以我们需要尽量将图块纹理分类及合并，在每个块层上仅使用一个纹理图来绘制。

(2) `CCTMXTiledMap`类不允许任何一层（`CCTMXLayer`）中没有任何元素，否则解析TMX时会报错。解决的办法是：要么将空图层删掉，要么在空图层上放置一个透明的图块，让用户无法察觉。

(3) 关于是否要将图块导出成外部图块（tileset），这一点需要看游戏本身的需求。如果你在外部图块中给图块配置了大量信息，比如怪物的属性和物品的功能等，这些对每个地图都是通用

的，这时导出成外部图块就省去了重复的复制工作。但由于Tiled编辑器不支持加密，外部图块的数据会被其他人看到。所以对于正式发布的游戏，不建议将影响游戏平衡性的数值配置在外部图块中。

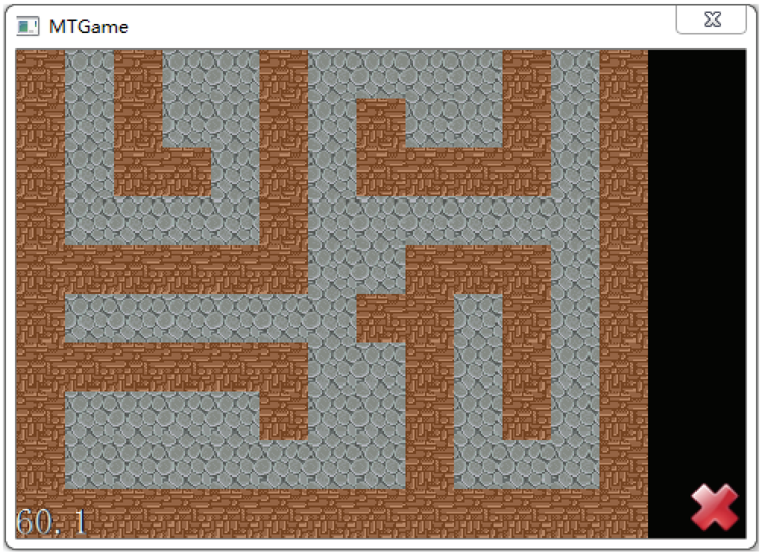


图10-8　在游戏中添加tmx地图

10.4　人物行走

前面一节中我们绘制了一张简单的地图，包括地板层和墙壁层。但略显单调，因为既没有人物，地图也无法拖动。下面我们就来给地图加上可以行走的勇士。

勇士可以被视为一个精灵（CCSprite），它包含了上、下、左、右4个方向的行走动作，如图10-9所示。



图10-9　勇士的纹理

将图10-9保存为hero.png，放到Resource目录下。我们首先来做人物向下移动一格的动画，也就是第一行的4幅图。首先将图片生成纹理，保存到全局的纹理缓存区：

```
CCTexture2D *heroTexture = CCTextureCache::sharedTextureCache()->addImage("hero.png");
```

然后用纹理创建4帧动画，并存放到CCMutableArray中：

```
CCSpriteFrame *frame0, *frame1, *frame2, *frame3;
//第二个参数表示显示区域的x, y, width, height
frame0 = CCSpriteFrame::frameWithTexture(heroTexture, cocos2d::CCRectMake(32*0, 32*0,
32, 32));
frame1 = CCSpriteFrame::frameWithTexture(heroTexture, cocos2d::CCRectMake(32*1, 32*0,
32, 32));
frame2 = CCSpriteFrame::frameWithTexture(heroTexture, cocos2d::CCRectMake(32*2, 32*0,
32, 32));
frame3 = CCSpriteFrame::frameWithTexture(heroTexture, cocos2d::CCRectMake(32*3, 32*0,
32, 32));
CCMutableArray<CCSpriteFrame *> *animFrames = new CCMutableArray<CCSpriteFrame*>(4);
animFrames->addObject(frame0);
animFrames->addObject(frame1);
animFrames->addObject(frame2);
animFrames->addObject(frame3);
```

根据4个动画帧生成CCAnimation对象：

```
CCAnimation *animation = new CCAnimation();
//0.2f表示每帧动画间的间隔
animation->initWithFrames(animFrames, 0.2f);
animFrames->release();
```

接下来，我们创建一个CCSprite用来显示勇士，可以使用Animation中的一帧来作为勇士静止时的画面：

```
//用frame0作为勇士的静态图
CCSprite *heroSprite = CCSprite::spriteWithSpriteFrame(frame0);
//暂时将勇士显示在(100, 100)处
heroSprite->setPosition(ccp(100, 100));
addChild(heroSprite);
```

最后，让heroSprite播放定义的动画，就有了行走的效果：

```
//根据动画模板创建动画
CCAnimate *animate = CCAnimate::actionWithAnimation(animation, false);
//创建不断重复的动画，并让heroSprite播放
heroSprite->runAction(CCRepeatForever::actionWithAction(animate));
```

现在看一下游戏的运行效果，画面左下角（坐标(100,100)处）出现了一个勇士在不停地走动，如图10-10所示。

下面我们来创建其他3个方向的动画，并把它们保存到相应的全局变量中。由于创建的步骤重复且复杂，我们将通用的代码提取成一个方法createAnimationByDirection(HeroDirection direction)。在HelloWorldScene.h中添加：

```
//数组保存勇士向4个方向行走的动画模板
CCAnimation **walkAnimations;
//根据方向创建动画
CCAnimation *createAnimationByDirection(int direction);
```



图10-10 勇士不停地行走

方法createAnimationByDirection的实现如下：

```
CCAnimation *HelloWorld::createAnimationByDirection(HeroDirection direction)
{
    CCTexture2D *heroTexture = CCTextureCache::sharedTextureCache()->addImage("hero.png");
    CCSpriteFrame *frame0, *frame1, *frame2, *frame3;
    //第二个参数表示显示区域的x, y, width, height，根据direction来确定显示的y坐标
    frame0 = CCSpriteFrame::frameWithTexture(heroTexture, cocos2d::CCRectMake(32*0,
    32*direction, 32, 32));
    frame1 = CCSpriteFrame::frameWithTexture(heroTexture, cocos2d::CCRectMake(32*1,
    32*direction, 32, 32));
    frame2 = CCSpriteFrame::frameWithTexture(heroTexture, cocos2d::CCRectMake(32*2,
    32*direction, 32, 32));
    frame3 = CCSpriteFrame::frameWithTexture(heroTexture, cocos2d::CCRectMake(32*3,
    32*direction, 32, 32));
    CCMutableArray<CCSpriteFrame *> *animFrames = new CCMutableArray<CCSpriteFrame *>(4);
    animFrames->addObject(frame0);
    animFrames->addObject(frame1);
    animFrames->addObject(frame2);
    animFrames->addObject(frame3);
    CCAnimation *animation = new CCAnimation();
    //0.2f表示每帧动画间的间隔
    animation->initWithFrames(animFrames, 0.2f);
    animFrames->release();
```

```
    return animation;
}
```

其中参数HeroDirection是枚举类型，可能传入的值在HelloWorldScene.h中定义：

```
typedef enum {
    kDown =  0,
    kLeft = 1,
    kRight = 2,
    kUp = 3,
} HeroDirection;
```

现在，可以在init方法中方便地创建4个方向的行走动画：

```
walkAnimation = new CCAnimation*[4];
walkAnimation[kDown] = createAnimationByDirection(kDown);
walkAnimation[kLeft] = createAnimationByDirection(kLeft);
walkAnimation[kRight] = createAnimationByDirection(kRight);
walkAnimation[kUp] = createAnimationByDirection(kUp);
```

还需要稍微修改heroSprite的初始化方法，使用downAnimation的第1帧作为勇士静态图：

```
CCSprite *heroSprite = CCSprite::spriteWithSpriteFrame(walkAnimation[kDown]->
getFrames()->getObjectAtIndex(0));
```

接下来我们需要加入一些操作，控制勇士的行走方向。最方便的是添加4个按钮，分别对应上下左右4个动作：

```
CCMenuItem *down = CCMenuItemFont::itemFromString("down", this, menu_selector
(HelloWorld::menuCallBackMove));
CCMenuItem *left = CCMenuItemFont::itemFromString("left", this, menu_selector
(HelloWorld::menuCallBackMove) );
CCMenuItem *right = CCMenuItemFont::itemFromString("right", this, menu_selector
(HelloWorld::menuCallBackMove) );
CCMenuItem *up = CCMenuItemFont::itemFromString("up", this, menu_selector
(HelloWorld::menuCallBackMove) );
CCMenu *menu = CCMenu::menuWithItems(down, left, right, up, NULL);
//为了方便查找，给每个menuItem设置Tag
down->setTag(kDown);
left->setTag(kLeft);
right->setTag(kRight);
up->setTag(kUp);
//菜单项按间距水平排列
menu->alignItemsHorizontallyWithPadding(50);
addChild(menu);
```

在回调函数menuCallBackMove方法中，我们根据sender的Tag值判断是哪个按钮，来执行对应的操作：

```
void HelloWorld::menuCallBackMove(CCObject *pSender)
{
    CCNode *node = (CCNode *) pSender;
    //按钮的Tag就是需要行走的方向
```

```
    int targetDirection = node->getTag();
    CCAction *action = CCSequence::actions(
            CCAnimate::actionWithAnimation(walkAnimation[targetDirection], false),
            //把方向信息传递给onWalkDone方法
            CCCallFuncND::actionWithTarget(this,
            callfuncND_selector(HelloWorld::onWalkDone), (void*)targetDirection),
            NULL);
    heroSprite->runAction(action);
}
```

现在运行代码，依次点击在屏幕中央的4个按钮，发现勇者会向不同方向行走。但是奇怪的是，在行走动作播放结束后，勇者总是会恢复到初始设置画面，也就是downAnimation的第1帧。这不是我们期望的结果，勇者在行走结束后，应该保持画面为行走方向状态。为此，需要在行走动画播放结束后，根据朝向设置勇者对应的静态画面。我们增加了一个方法setFacingDirection (HeroDirection direction)：

```
void HelloWorld::setFaceDirection(HeroDirection direction)
{
    heroSprite->setTextureRect(CCRectMake(0,32*direction,32,32));
}
```

调用这个方法的时机是在行走动画结束后，因此要对动画增加结束时的回调函数，修改menuCallBackMove方法。注意，这里使用了CCCallFuncND的回调函数，它包括两个参数，第一个参数是发起回调的CCNode节点，第二个参数是传递的数值。我们将行走的方向作为第二个参数传给onWalkDone方法，使其能够知道如何设置勇士停止后的静态图。

```
void HelloWorld::menuCallBackMove(CCObject *pSender)
{
    CCNode *node = (CCNode *) pSender;
    int targetDirection = node->getTag();
    CCAction *action = CCSequence::actions(
      CCAnimate::actionWithAnimation(walkAnimation[targetDirection],
      false),
      CCCallFuncND::actionWithTarget(this,
      callfuncND_selector(HelloWorld::onWalkDone), (void*)targetDirection),
      NULL);
    heroSprite->runAction(action);
}
```

在方法onWalkDone里面，只需要简单地调用setFaceDirection即可：

```
void HelloWorld::onWalkDone(CCNode *pTarget, void *data)
{
    //将void*先转换为int，再从int转换到枚举类型
    int direction = (int) data;
    setFacingDirection((HeroDirection)direction);
}
```

现在让我们再次运行代码，体验一下目前的效果。屏幕中央会出现4个文字按钮，点击不同

的按钮，勇士会向上下左右4个方向行走，如图10-11所示。



图10-11　点击按钮操作勇士行走

等等，我们是不是忘记了什么东西？仔细检查一下代码，看看是否会有内存泄漏问题？没错，我们在创建动画的过程中new了一个CCAnimation*的数组，并且创建了4个动画模板，但直到程序退出都没有销毁它们。现在我们在HelloWorld的析构函数中把它们销毁掉。这里我们使用了cocos2d-x内置的两个宏（CC_SAFE_RELEASE和CC_SAFE_DELETE_ARRAY），可以方便地释放CCObject型对象和数组。还有其他一些常用宏请参考CCPlatformMacros.h。

```
HelloWorld::~HelloWorld(void)
{
    for (int i = 0; i < 4; i++)
    {
        //释放数组中元素
        CC_SAFE_RELEASE(walkAnimation[i])
    }
    //释放数组本身
    CC_SAFE_DELETE_ARRAY(walkAnimation);
}
```

到目前为止，勇士只能悲剧地在原地打转，现在我们就让它在地图上动起来。接下来将会用到两个动作里的重要方法，即CCSpawn和CCMoveBy，其中CCSpawn可以实现多个动作同时执行，其参数表是可变的。

```
CCFiniteTimeAction *CCSpawn::actions(cocos2d::CCFiniteTimeAction *pAction1, ...)
```

CCMoveBy可以使精灵在指定时间内移动指定的距离：

```
CCMoveBy *CCMoveBy::actionWithDuration(cocos2d::ccTime duration, cocos2d::CCPoint
position)
```

关于以上两个类，大家可以去test示例中的ActionTest部分查看更多的使用方法。有了这两个类，只需要对menuCallBackMove方法稍作修改即可让勇士在地图上动起来。首先根据方向计算移动距离，我们规定每次走地图上的一小格，也就是32个像素，然后将行走动画和moveBy的动作同时执行。具体代码如下：

```
void HelloWorld::menuCallBackMove(CCObject *pSender)
{
    CCNode *node = (CCNode *) pSender;
    //按钮的Tag就是需要行走的方向
    int targetDirection = node->getTag();
    //移动的距离
    CCPoint moveByPosition;
    //根据方向计算移动的距离
    switch (targetDirection)
    {
    case kDown:
        moveByPosition = ccp(0, -32);
        break;
    case kLeft:
        moveByPosition = ccp(-32, 0);
        break;
    case kRight:
        moveByPosition = ccp(32, 0);
        break;
    case kUp:
        moveByPosition = ccp(0, 32);
        break;
    }
    //利用CCSpawn将行走动画和移动同时执行
    CCAction *action = CCSequence::actions(
            CCSpawn::actions(
            CCAnimate::actionWithAnimation(walkAnimation[targetDirection],
            false),
            CCMoveBy::actionWithDuration(0.28f, moveByPosition),
            NULL),
            //把方向信息传递给onWalkDone方法
            CCCallFuncND::actionWithTarget(this,
            callfuncND_selector(HelloWorld::onWalkDone), (void*)targetDirection),
            NULL);
    heroSprite->runAction(action);
}
```

好了，现在再次运行工程，点击上下左右方向按钮，勇士终于可以顺畅地在地图上漫步了。不过，你可能发现了一个小bug：如果很快地点击按钮，在勇士没有走完一格的时候就输入下一个移动指令，它就会停止当前的移动，直接执行新的指令。所以要保证勇士在走完一格之前，不接受新的移动指令。我们增加了一个全局的bool变量isHeroWalking（是否正在移动）。在勇士开始移动的时候将其设置为true，移动完成的时候将其设置为false。若在回调函数menuCallBackMove开始时判断isHeroWalking为true，则直接return，不执行移动命令：

```
void HelloWorld::menuCallBackMove(CCObject *pSender)
{
```

```
    //勇士正在移动中，不接受新的指令
    if (isHeroWalking)
        return;
    //移动操作
    ......
    //勇士开始移动，设置标识为true
    isHeroWalking = true;
}

void HelloWorld::onWalkDone(CCNode *pTarget, void* data)
{
    //移动完毕，设置表示为false
    isHeroWalking = false;
      //移动完毕操作
      ......
}
```

如果我们多体验一下目前的游戏，就会发现勇士走起路来怪怪的。没错，感觉像是悬浮在地图上，碰到墙没有停止，行走范围也没有在格子规定的范围内。下面的工作就是把勇士规整地贴合在地图内，让它不要乱跑。先从最简单的入手，假设初始时，我们想把勇士放在地图的左下角，如图10-12所示。



图10-12　将勇士放置在地图左下角

为了实现这个功能，首先要了解Tilemap的坐标系和cocos2d-x的坐标系。Tilemap的坐标系是以左上角为原点，而cocos2d-x坐标系（如图10-13所示）以左下角为原点。

在图10-12中，勇士所在的位置按照Tilemap的坐标系计算应为(1, 11)，那么对应的cocos2d-x的坐标是多少呢？只需要将*y*坐标反转，用Tilemap的最大高度减去当前*y*的高度，再乘以图块的宽高即可，计算规则如图10-14所示。

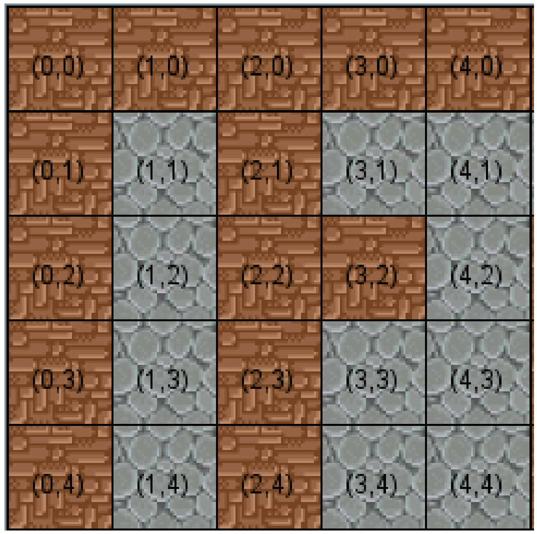


图10-13 Tilemap坐标系

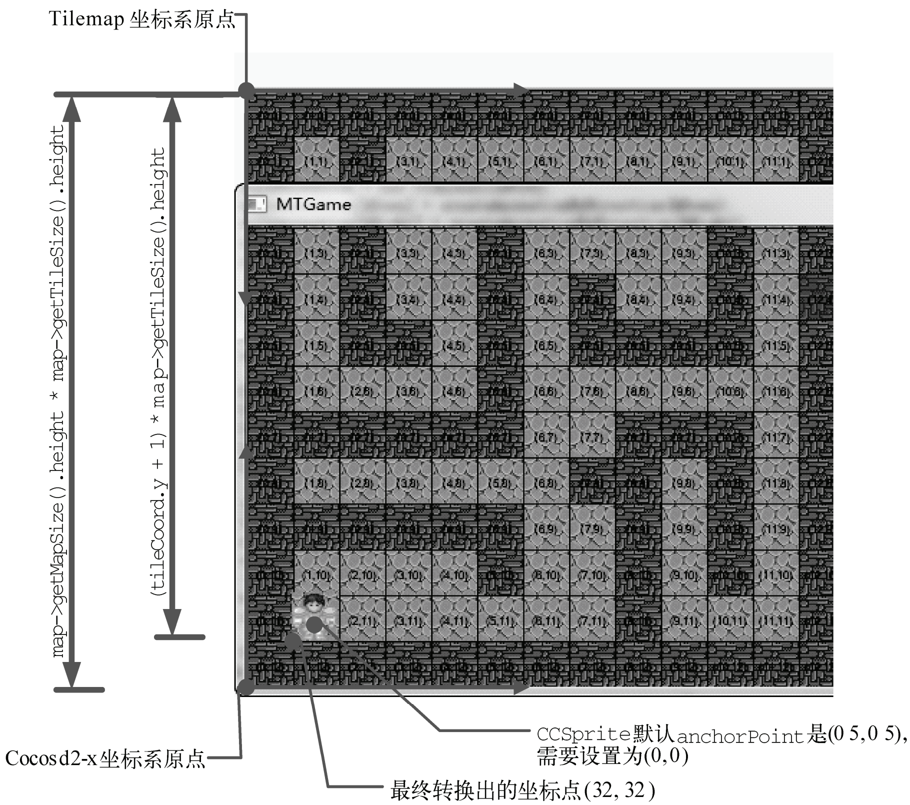


图10-14 Tilemap坐标系和cocos2d-x坐标系转换

在HelloWorldScene.cpp中增加一个方法positionForTileCoord，功能是实现Tilemap坐标系到cocos2d-x坐标系的转换，具体代码如下：

```
//从Tilemap坐标转换为cocos2d-x坐标
CCPoint HelloWorld::positionForTileCoord(CCPoint tileCoord)
{
    CCPoint pos =  ccp((tileCoord.x *map->getTileSize().width),
        ((map->getMapSize().height - tileCoord.y - 1) *map->getTileSize().height));
    return pos;
}
```

接下来修改勇士精灵的锚点为(0, 0)，起始位置设为(1, 11)：

```
heroSprite->setAnchorPoint(CCPointZero);
heroSprite->setPosition(positionForTileCoord(ccp(1, 11)));
```

这样就成功地将勇士放置在地图左下角了。接下来，我们为游戏加入场景滚动的效果。设想一下，随着勇士的移动，原本不在视野内的地图需要逐渐显示出来。为了便于理解，先只讨论*y*轴上的场景滚动。假设勇士已经移动到Tilemap的(1, 4)位置，对应cocos2d-x坐标为(32, 224)，如何计算出场景应该滚动多少距离？首先，将屏幕高度的1/2作为滚动的临界位置，*y*值小于1/2高度的不需要滚动，大于1/2的才开始滚动。为什么要把屏幕的1/2作为临界位置呢？因为这样可以保证场景在滚动时，勇士始终处于屏幕高度的1/2处，这样的视觉效果最佳。当然你也可以使用1/3或2/3高度作为临界点，但这样就会有一种不对称的感觉。现在计算出了屏幕的一半高度是320/2=160像素，而勇士的*y*值为224，那么场景需要滚动的距离就是224−(320/2)=64像素，如图10-15所示。此外，还有以下几个地方需要注意。

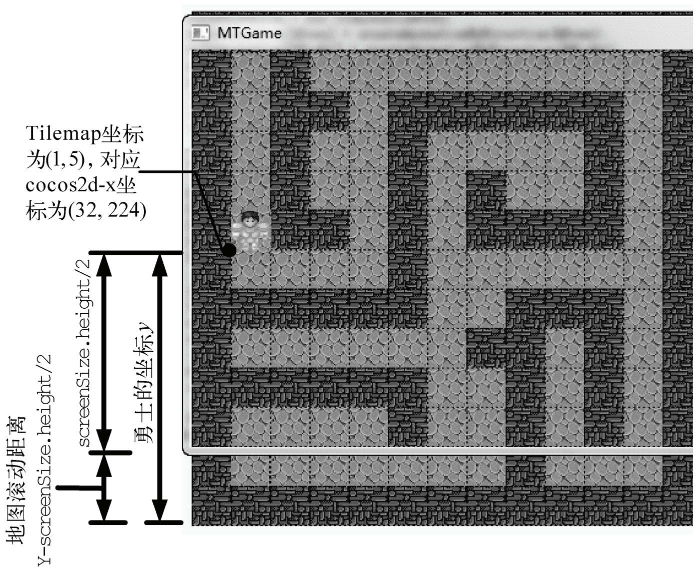


图10-15 计算场景滚动距离

(1) 如果地图总宽/高小于屏幕的宽/高，那么直接可以断定不需要滚动。

(2) 场景滚动的最大距离不能超过地图总宽高减去屏幕宽高的1/2，否则在勇士走到地图边缘时，场景继续滚动，会造成屏幕周围显示黑边。

(3) 这里使用的“移动”是场景移动，而不是单纯的地图移动。实际上，我们需要连人带地图一起移动！待会运行游戏的时候可以观察一下，当勇者在超过屏幕1/2处行走时，虽然给人感觉上是在走动，但实际相对屏幕的位置没有发生变化，仍然在屏幕1/2处。

好了，现在已经知道了场景滚动的原理，下面让我们用代码实现它！首先在HelloWorld-Scene.cpp中添加一个方法setSceneScrollPosition。它有一个参数，是勇士当前在cocos2d-x坐标系内的位置。此方法可以将场景移动到相应位置。

```
//传入勇士当前位置信息，将场景移动到相应位置
void GameLayer::setSceneScrollPosition(CCPoint position)
{
    //获取屏幕尺寸
    CCSize screenSize = CCDirector::sharedDirector()->getWinSize();
    //计算Tilemap的宽高，单位是像素
    CCSize mapSizeInPixel = CCSizeMake(map->getMapSize().width * map->getTileSize().width,
        map->getMapSize().height * map->getTileSize().height);
    //取勇士当前x坐标和屏幕中点x的最大值，如果勇士的x值较大，则会滚动
    float x = MAX(position.x, screenSize.width / 2.0f);
    float y = MAX(position.y, screenSize.height / 2.0f);
    //地图总宽度大于屏幕宽度的时候才有可能滚动
    if (mapSizeInPixel.width  > screenSize.width)
    {
        //场景滚动的最大距离不能超过地图总宽减去屏幕宽的/2
        x = MIN(x, mapSizeInPixel.width - screenSize.width / 2.0f);
    }
    if (mapSizeInPixel.height > screenSize.height)
    {
        y = MIN(y, mapSizeInPixel.height - screenSize.height / 2.0f);
    }
    //勇士的实际位置
    CCPoint heroPosition = ccp(x, y);
    //屏幕中点位置
    CCPoint screenCenter = ccp(screenSize.width/2.0f, screenSize.height/2.0f);
    //计算勇士实际位置和重点位置的距离
    CCPoint scrollPosition = ccpSub(screenCenter, heroPosition);
    //将场景移动到相应位置
    this->setPosition(scrollPosition);
    CCLog("%f,%f", scrollPosition.x, scrollPosition.y);
}
```

那么，什么时候使用setSceneScrollPosition方法呢？是在onWalkDone函数中吗？可以尝试一下，发现场景会在移动完成后，突然跳到指定位置，这种效果体验太差。理论上只要勇士发生了位置变化，就需要调用此函数更新场景位置。所以不能单纯地在onWalkDone中更新场景位置。那如何来监听勇士位置变化的事件呢？前面我们使用了CCMoveBy来进行勇士的移动，

在其执行过程中并不受控制，因此无法直接监听到勇士移动的事件。所以，只能在游戏的每帧里来做这件事情了。我们新建一个schedule_selector：HelloWorld::update，设置其调用间隔为每帧，在里面实现对场景位置的更新。然后在场景初始化的时候启动定时器，并在析构函数里销毁定时器。

```
void HelloWorld::update(ccTime dt)
{
    //如果勇士不在行走状态，不需要更新场景位置
    if (isHeroWalking)
    {
        setSceneScrollPosition(heroSprite->getPosition());
    }
}

//在场景init的时候初始化一个定时器
bool HelloWorld::init()
{
    bool bRet = false;
    do
    {
        //若干初始化操作
        ……
        //设置定时器为每帧调用
        schedule(schedule_selector(HelloWorld::update));
        bRet = true;
    } while (0);
    return bRet;
}

//注意在析构函数里销毁定时器
HelloWorld::~HelloWorld(void)
{
    //若干清除操作
    ……
    //直接清除所有定时器
    this->unscheduleAllSelectors();
}
```

好了，现在编译运行程序，点击方向按钮，勇士可以随意地在地图内穿梭了。不过仍然有几个潜在的bug不知道你有没有注意到。

第一个bug是地图上各个图块之间有一些细密的间隔。特别是在场景移动时更加明显。我们将游戏截图放大，可以清楚地看到这些细线。这个问题可以通过设置ccConfig.h中的CC_FIX_ARTIFACTS_BY_STRECHING_TEXEL宏来解决。这个值默认是0，将其设置为1以后，编译运行。对比效果如图10-16所示。解决的大致原理是贴图时只适用99%的纹理，将纹理的四边形的范围缩小一圈：

```
GLfloat left = (rect.origin.x*2+1) / (wide*2);
```

```
GLfloat bottom = (rect.origin.y*2+1) / (high*2);
GLfloat right = left + (rect.size.width*2-2) / (wide*2);
GLfloat top = bottom + (rect.size.height*2-2) / (high*2);
```

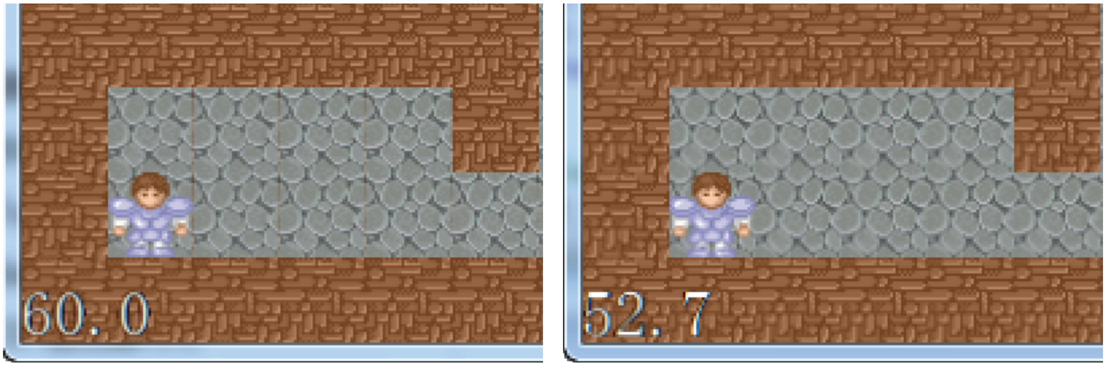


图10-16　CC_FIX_ARTIFACTS_BY_STRECHING_TEXEL开启前后效果对比

第二个bug是在场景发生滚动的过程中地图会抖动。我们可以通过对Tilemap的每个图层的纹理开启抗锯齿效果来解决这个问题。具体代码如下：

```
CCArray *pChildrenArray = map->getChildren();
CCSpriteBatchNode *child = NULL;
CCObject *pObject = NULL;
//遍历Tilemap的所有图层
CCARRAY_FOREACH(pChildrenArray, pObject)
{
    child = (CCSpriteBatchNode*)pObject;
    if(!child)
       break;
    //给图层的纹理开启抗锯齿
    child->getTexture()->setAntiAliasTexParameters();
}
```

第三个bug是一个更隐蔽的bug。让我们给setSceneScrollPosition加上log，跟踪一下场景移动的距离。在方法的最后加上以下代码：

```
CCLog("scene position: %f,%f", scrollPosition.x, scrollPosition.y);
```

然后运行程序，控制勇士移动，使场景发生滚动，观察log输出：

scene position: 0.000000, 0.000000

scene position: 0.000000, –1.942856

scene position: 0.000000, –3.885712

scene position: 0.000000, –5.714279

scene position: 0.000000, –7.657150

scene position: 0.000000, –9.600006

scene position: 0.000000, −11.428574

scene position: 0.000000, −13.371429

scene position: 0.000000, −15.314285

scene position: 0.000000, −17.142853

scene position: 0.000000, −19.085724

scene position: 0.000000, −21.028580

scene position: 0.000000, −22.857147

scene position: 0.000000, −24.800003

scene position: 0.000000, −26.742859

scene position: 0.000000, −28.571426

scene position: 0.000000, −30.514282

问题出现了，场景最终只被移动到了(0, −30.514282)位置，而预期位置是(0, −32)。什么地方出差错了呢？按道理update方法会一直被调用，只有判断isHeroWalking为false的情况下才不会去更新场景位置。也就是说最后一次移动后，isHeroWalking立刻被置为false，导致update方法没有去调用setSceneScrollPosition。如果猜测成立的话，解决这个问题只需要在isHeroWalking被置为false之后，手动调用一次setSceneScrollPosition即可。尝试修改onWalkDone方法：

```
void HelloWorld::onWalkDone(CCNode *pTarget, void *data)
{
    int direction = (int) data;
    setFaceDirection((HeroDirection)direction);
    isHeroWalking = false;
    //手动调用更新场景的方法
    setSceneScrollPosition(heroSprite->getPosition());
}
```

修改完毕后，运行观察log输出，发现一切正常。场景可以被正确地移动到指定位置，即(0, −32)处。

scene position: 0.000000, 0.000000

scene position: 0.000000, −1.942856

scene position: 0.000000, −3.885712

scene position: 0.000000, −5.714279

scene position: 0.000000, −7.657150

scene position: 0.000000, −9.600006

scene position: 0.000000, −11.428574

scene position: 0.000000, −13.371429

scene position: 0.000000, −15.314285

scene position: 0.000000, –17.142853
scene position: 0.000000, –19.085724
scene position: 0.000000, –21.028580
scene position: 0.000000, –22.857147
scene position: 0.000000, –24.800003
scene position: 0.000000, –26.742859
scene position: 0.000000, –28.571426
scene position: 0.000000, –30.514282
scene position: 0.000000, –32.000000

好了，到目前为止，我们基本实现了勇士在地图上移动的效果。这一节的内容比较长，让我们来总结回顾一下具体步骤吧。

(1) 我们通过`CCSpriteFrame`定义了勇士上、下、左、右4个方向的行走动画，并创建出动画模板存放在数组里方便行走时调用。接着根据动画的某一帧创建了勇士的精灵对象。

(2) 创建了4个方向的文字按钮，点击后可以让勇士播放相应的行走动画。在按钮的回调函数中，可以通过发起者的Tag值判断是哪个按钮被按下。

(3) 用`CCSequence`和`CCCallFuncND`注册了回调函数，可以监听行走的结束事件，设置勇士的朝向和行走状态标识。

(4) 使用了`CCSpawn`和`CCMoveBy`让勇士在播放行走动画的同时移动。

(5) 为了让勇士贴合在地图内，我们研究了如何将Tilemap坐标系转换为cocos2d-x坐标系。

(6) 最后，我们实现了场景的滚动效果。根据勇士的坐标计算出场景需要滚动的距离，并注册了一个每帧触发的定时器更新场景位置。

10.5 碰撞检测

前面已经初步实现了勇士在地图内的移动。这一节我们就开始加入碰撞检测，让勇士在碰到墙壁之后无法行走，到地图边缘时也不会走出去，遇到敌人的时候会战斗等。

我们先来实现墙壁的碰撞逻辑。大家可以先思考一下，如何判断勇士撞到墙了？相信你已经有了一些思路了：可以在`menuCallBackMove`方法中判断目标地点的图块类型，如果是墙壁的话，就不让勇士移动。没错，这就是我们将要采用的方法。下面我们看一下如何用代码实现这个逻辑。

(1) 先熟悉一下`CCTMXLayer`中的一个重要方法`tileGIDAt(CCPoint tileCoordinate)`。此方法可以根据传入的Tilemap坐标，返回坐标位置的图块编号，如果该位置没有图块，则返回0。有了这个方法，就可以方便地判断目标位置是否有墙壁了。现在只需要将勇士的cocos2d-x坐标转换为Tilemap坐标即可。转换方法如下，在计算*y*坐标时，用最大的*y*值减去当前*y*值即可。

```
//从cocos2d-x坐标转换为Tilemap坐标
```

```
CCPoint HelloWorld::tileCoordForPosition(CCPoint position)
{
    int x = position.x / map->getTileSize().width;
    int y = (((map->getMapSize().height - 1) * map->getTileSize().height) - position.y)
    / map->getTileSize().height;
    return ccp(x, y);
}
```

(2) 在HelloWorldScene.h头文件里定义几个碰撞类型：

```
typedef enum
{
    kNone = 1,//可以通行
    kWall,//墙
    kEnemy,//敌人
} CollisionType;//碰撞类型
```

(3) 添加一个方法checkCollision（用于碰撞检测）。注意：在获取图块ID之前，先判断目标位置是否在地图范围内，如果超出了地图范围，则返回kWall类型，阻止勇士移动。

```
//判断碰撞类型
CollisionType HelloWorld::checkCollision(CCPoint heroPosition)
{
    //cocos2d-x坐标转换为Tilemap坐标
    CCPoint tileCoord = tileCoordForPosition(heroPosition);
    //如果勇士坐标超过地图边界，返回kWall类型，阻止其移动
    if (heroPosition.x < 0 || tileCoord.x > map->getMapSize().width - 1 || tileCoord.y
        < 0 || tileCoord.y > map->getMapSize().height - 1)
        return kWall;
    //获取当前坐标位置的图块ID
    int tileGid = map->layerNamed("wall")->tileGIDAt(tileCoord);
    //如果图块ID不为，表示有墙
    if (tileGid) {
        return kWall;
    }
    //可以通行
    return kNone;
}
```

(4) 在menuCallBackMove这个回调方法中调用checkCollision，检测碰撞。如果碰撞类型是墙壁，只需要设置勇士的朝向；如果无障碍，则开始行走。

```
void HelloWorld::menuCallBackMove(CCObject* pSender)
{
    //根据方向计算移动的距离
    ……
    case kUp:
        moveByPosition = ccp(0, 32);
        break;
    }
    //计算目标坐标，用当前勇士坐标加上移动距离
    CCPoint targetPosition = ccpAdd(heroSprite->getPosition(), moveByPosition);
```

```
    //调用checkCollision检测碰撞类型，如果是墙壁，则只需要设置勇士的朝向
    if (checkCollision(targetPosition) == kWall)
    {
        setFaceDirection((HeroDirection)targetDirection);
        return;
    }
    //利用CCSpawn将行走动画和移动同时执行
    ……
}
```

现在编译运行程序，控制勇士在地图上行走，当遇到墙的时候，勇士不会再穿墙而过了，而是乖乖地站在原地不动。

10.6 总结

经过本章的学习，我们已经完成了一个简单的魔塔游戏，能够显示基于TMX的地图，能够显示精灵（包括勇士和怪物），能够检测元素之间的碰撞。在接下来的一章里，我们会对游戏进行进一步优化，给魔塔添加更多的元素和更多的游戏规则。

第11章

"魔塔"案例之高级篇

在上一章里，我们主要介绍了开发魔塔游戏的一些基础知识，包括地图绘制、人物行走和碰撞检测等。本章将进一步介绍开发魔塔游戏的高级知识，即添加更多的元素和更多的游戏规则等。

11.1 重构代码

到目前为止，我们头疼地发现所有方法都是在同一个类（HelloWorldScene）里实现的。随着新功能的增加，代码变得异常臃肿，阅读和修改也变得越发困难。为了以后的可持续发展，让我们果断暂停新功能的开发，开始对老代码进行重构。

11.1.1 分离场景和图层

首先，我们对HelloWorldScene既包含了场景（CCScene）又包含了层（CCLayer）的做法表示不满：从名字来看，它是个场景（CCScene），可层（CCLayer）部分它也包含进去了。于是干脆将它分离成两个类：GameScene和GameLayer。

- GameScene类继承于CCScene，主要负责游戏主场景的创建，它可以与以后游戏中各种场景进行切换，比如主菜单场景。同时，它负责创建游戏主场景内的各个CCLayer，包括地图层、控制按钮层、对话框层和特效层等。
- GameLayer继承CCLayer，负责绘制游戏主界面的内容，包括地图、勇士和战斗效果等。

好了，现在来创建GameScene和GameLayer这两个类。

在这之前，首先创建一个公用文件MTGame.h，里面包含了项目中需要公用的类的头文件。这样做的好处是管理方便，所有头文件一目了然。缺点是你修改某个类的定义的时候，会引起其他所有类的重新编译。对于我们目前的游戏来说，这点牺牲还是值得的。如果你的游戏复杂到了一定程度，编译需要很长时间，那么还是老老实实地分别管理头文件比较好。言归正传，MTGame.h文件如下：

```
#ifndef __MTGAME_H__
#define __MTGAME_H__
```

```
#include "cocos2d.h"
#include "GameLayer.h"
#include "GameScene.h"

#endif
```

GameScene中需要提供一个静态方法showGameScene，用来生成一个新的游戏主界面实例，头文件代码如下：

```
#ifndef __GAME_SCENE_H__
#define __GAME_SCENE_H__
//包含公用头文件
#include "MTGame.h"
//使用cocos2d命名空间
using namespace cocos2d;
//GameScene继承CCScene
class GameScene : public CCScene
{
public:
    GameScene(void);
    ~GameScene(void);
    //静态方法用于创建新的游戏主界面的实例
    static CCScene *playNewGame();
    //初始化函数
    virtual bool init();
    //Scene的静态创建方法
    SCENE_NODE_FUNC(GameScene);};

#endif
```

GameScene的代码文件（cpp）如下，首先会使用CCScene的node方法创建一个实例，然后调用GameLayer的node方法创建layer的实例，并将其加入到scene当中：

```
#include "GameScene.h"

GameScene::GameScene(void)
{
}

GameScene::~GameScene(void)
{
}

bool GameScene::init()
{
    //新建一个GameLayer实例
    GameLayer *gamelayer = GameLayer::node();
    //将GameLayer实例添加到场景中
    this->addChild(gamelayer, kGameLayer, kGameLayer);
    //新建一个ControlLayer实例
    ControlLayer *controlLayer = ControlLayer::node();
    //将ControlLayer实例添加到场景中
    this->addChild(controlLayer, kControlLayer, kControlLayer);
```

```
    return true;
}

CCScene *GameScene::playNewGame()
{
    GameScene *scene = NULL;
    do
    {
        scene = GameScene::node();
        CC_BREAK_IF(! scene);
    } while (0);
    return scene;
}
```

对于GameLayer类，我们先把原来HelloWorldScene.cpp中HelloWorld类中相关的代码复制出来，重命名为GameLayer即可。继续修改AppDelegate中创建场景的部分：

```
//创建游戏主界面
CCScene *pScene = GameScene::playNewGame();
//运行场景
pDirector->runWithScene(pScene);
```

11.1.2 分离游戏对象

现在我们已经将HelloWorldScene.cpp中的代码分离成GameScene和GameLayer两部分。可以看出GameScene类的功能已经相对单一了，但GameLayer类仍然内容驳杂。下面来想办法继续拆分GameLayer类。目前游戏主界面包含3个对象，即地图、勇士和控制按钮（关闭按钮和方向按钮）。因此可以将GameLayer中的方法按这3大类进行划分。

(1) 地图类负责TMX地图的各项初始化操作以及Tilemap坐标系和cocos2d-x坐标系之间的转换。

(2) 勇士类负责行走动画的创建、在地图上移动以及碰撞检测。

(3) 控制层则包括关闭按钮和方向按钮的逻辑。

1. 游戏地图类

先分离地图对象。我们的游戏地图类在CCTMXTiledMap功能基础上，添加了额外的初始化操作：包括方便地获取各个层、开启纹理抗锯齿、以后可能会有的敌人以及NPC（Non-Player-Controlled Character）属性的读取。另外，Tilemap和cocos2d-x坐标系的转换以及碰撞检测也可以放在地图类中。现在，新建一个GameMap类，继承自CCTMXTiledMap，然后将需要实现的方法写在头文件里，具体代码如下：

```
#ifndef __GAME_MAP_H__
#define __GAME_MAP_H__

#include "MTGame.h"

typedef enum
```

```
{
    kNone = 1,//可以通行
    kWall,//墙
    kEnemy,//敌人
} CollisionType;//碰撞类型

using namespace cocos2d;

//继承自CCTMXTIledMap
class GameMap : public cocos2d::CCTMXTiledMap
{
    //只读变量，获取地板层和墙壁层
    CC_PROPERTY_READONLY(CCTMXLayer*, floorLayer, FloorLayer);
    CC_PROPERTY_READONLY(CCTMXLayer*, wallLayer, WallLayer);
public:
    GameMap(void);
    ~GameMap(void);
    //静态方法，用于生成GameMap实例
    static GameMap *gameMapWithTMXFile(const char *tmxFile);
    //TiledMap和cocos2d-x坐标系相互转换的方法
    CCPoint tileCoordForPosition(CCPoint position);
    CCPoint positionForTileCoord(CCPoint tileCoord);
protected:
    //TliedMap额外的初始化方法
    void extraInit();
    //开启各图层的纹理抗锯齿
    void enableAnitiAliasForEachLayer();
};

#endif
```

在类的开始我们使用了CC_PROPERTY_READONLY宏，这是cocos2d-x引擎内嵌的宏，可以方便地声明变量的get和set方法，具体用法参见CCPlateformMacros.h。这里声明了两个protected的CCTMXLayer指针类型的变量，用来获取地板层和墙壁层。在cpp文件中，需要自己实现getter方法：

```
//返回地板层
CCTMXLayer *GameMap::getFloorLayer()
{
    return floorLayer;
}

//返回墙壁层
CCTMXLayer *GameMap::getWallLayer()
{
    return wallLayer;
}
```

静态方法gameMapWithTMXFile用于生成GameMap实例。它遵循了cocos2d-x生成对象的规范：new一个对象，调用其init方法，设置对象为autorelease。具体代码如下：

```
//静态方法，用于生成GameMap实例
GameMap *GameMap::gameMapWithTMXFile(const char *tmxFile)
{
    //new一个对象
    GameMap *pRet = new GameMap();
    //调用init方法
    if (pRet->initWithTMXFile(tmxFile))
    {
        //调用额外的init方法
        pRet->extraInit();
        //将实例放入autorelease池，统一由引擎控制对象的生命周期
        pRet->autorelease();
        return pRet;
    }
    CC_SAFE_DELETE(pRet);
    return NULL;
}
```

在gameMapWithTMXFile中除了调用CCTMXTiledMap自有的initWithTMXFile方法以外，还调用了我们自己定义的extraInit方法，用于进行一些额外的初始化操作。

```
//TliedMap额外的初始化方法
void GameMap::extraInit()
{
    //开启各个图层的纹理抗锯齿
    enableAnitiAliasForEachLayer();
    //初始化地板层和墙壁层对象
    floorLayer = this->layerNamed("floor");
    wallLayer = this->layerNamed("wall");
}
```

enableAnitiAliasForEachLayer方法用于开启TiledMap中所有层的纹理抗锯齿效果：

```
void GameMap::enableAnitiAliasForEachLayer()
{
    CCArray *pChildrenArray = this->getChildren();
    CCSpriteBatchNode *child = NULL;
    CCObject *pObject = NULL;
    //遍历Tilemap的所有图层
    CCARRAY_FOREACH(pChildrenArray, pObject)
    {
        child = (CCSpriteBatchNode*)pObject;
        if(!child)
            break;
        //给图层的纹理开启抗锯齿
        child->getTexture()->setAntiAliasTexParameters();
    }
}
```

tileCoordForPosition、positionForTileCoord和checkCollision方法基本与前面HelloWorldScene中的实现一致，这里不再赘述。最后在GameLayer中，只需要调用GameMap

的gameMapWithTMXFile方法就可以很方便地创建游戏地图了，具体代码如下：

```
map = GameMap::gameMapWithTMXFile("0.tmx");
addChild(map);
```

2. 勇士类

接下来我们开始分离勇士类，暂命名为Hero类。Hero类中应包括用于显示的精灵、各种属性（攻击、防御和生命值）和勇士移动的方法等。

首先，请大家思考一个问题：Hero类是否应该继承于CCSprite？我们上节实现的勇士移动，仅仅使用了一个CCSprite，那么现在要往里面添加额外的方法，继承自CCSprite应该就可以满足了。这种考虑没错，确实就目前要实现的功能来说，这个方法是可行的。但是如果能看得稍微远一点，就会发现这个方案缺乏灵活性。设想一下，如果我们想在勇士的头顶上加一个用于显示生命值的血条，要求它能随勇士移动而移动，难道需要写一个update方法，不停地刷新生命条的坐标吗？这显然不现实，也不合理。同理还有勇士的装备和光环buff的技能效果等。所以我们让Hero类继承CCNode，然后将用于显示的精灵作为孩子节点添加进去，而游戏主界面将Hero整体作为一个CCNode保存为孩子节点。这样针对Hero主体的缩放、移动、执行动作、显示和隐藏等操作都会影响到Hero类内部的所有显示节点。下面来看下Hero类头文件的定义：

```
#ifndef __HERO_H__
#define __HERO_H__

#include "MTGame.h"

using namespace cocos2d;

//勇士类继承自CCNode
class Hero : public cocos2d::CCNode
{
public:
    Hero(void);
    ~Hero(void);
    //静态方法，用于创建勇士实例
    static Hero *heroWithinLayer();
    //初始化方法
    bool heroInit();
    //用于显示勇士形象的精灵
    CCSprite *heroSprite;
    //标识勇士是否在移动状态
    bool isHeroMoving;
    //让勇士向指定方向移动一格
    void move(HeroDirection direction);
    //移动完成后的回调函数
    void onMoveDone(CCNode *pTarget, void* data);
    //设置勇士朝向
    void setFaceDirection(HeroDirection direction);
```

```
};

#endif
```

静态方法heroWithinLayer用于创建勇士实例，仍然遵循cocos2d-x的对象创建方式。于GameMap类中的gameMapWithTMXFile方法作用类似，相信大家对此都已经比较了解。具体代码如下：

```
//静态方法，用于生成Hero实例
Hero *Hero::heroWithinLayer()
{
    //new一个对象
    Hero *pRet = new Hero();
    //调用init方法
    if (pRet && pRet->heroInit())
    {
        //将实例放入autorelease池，统一由引擎控制对象的生命周期
        pRet->autorelease();
        return pRet;
    }
    CC_SAFE_DELETE(pRet)
    return NULL;
}
```

heroInit是用来初始化Hero类的，主要完成heroSprite的新建、添加、设置锚点以及一些标志位的赋值等。具体代码如下。注意，CCSprite::spriteWithSpriteFrame方法并没有给完全，暂时先放在这里，因为我们将要在下一小节中引入一个新类来管理动画。

```
bool Hero::heroInit()
{
    //根据向下行走动画的第一帧创建精灵
    heroSprite = CCSprite::spriteWithSpriteFrame(/*根据指定动画帧创建对象*/);
    //设置锚点
    heroSprite->setAnchorPoint(CCPointZero);
    //将用于显示的heroSprite加到自己的节点下
    this->addChild(heroSprite);
    //一开始不处于move状态
    isHeroMoving = false;
    return true;
}
```

Hero的move方法除了照搬HelloWorldScene中的menuCallBackMove以外，还需要做一些修改。原来我们为了对单个CCSprite实现移动和播放动画同时进行的效果，使用了CCSpawn。现在由于Hero类可能会承载多个精灵，不能单纯地只移动heroSprite一个精灵，而是要将作为CCNode的Hero主体类整体进行移动。所以heroSprite仅需要播放行走动画，而Hero类进行位移动作。注意，原来heroSprite->getPosition()的地方都需要改成this->getPosition()。另外，为了保证Hero是在移动到目标位置后再执行onMoveDone，我们将其与Hero节点本身的CCMoveBy动作拼接成为一个串行操作。move方法的部分代码如下：

```
void Hero::move(HeroDirection direction)
{
    if (isHeroMoving)
        return;
    //若干操作
    ……
    //计算目标坐标，用当前勇士坐标加上移动距离
    CCPoint targetPosition = ccpAdd(this->getPosition(), moveByPosition);
    //计算目标坐标，用当前勇士坐标加上移动距离
    CCPoint targetPosition = ccpAdd(this->getPosition(), moveByPosition);
    //调用checkCollision检测碰撞类型，如果是墙壁，则只需要设置勇士的朝向
    if (checkCollision(targetPosition) == kWall)
    {
        setFaceDirection((HeroDirection)direction);
        return;
    }
    //heroSprite仅播放行走动画
    heroSprite->runAction(/*获取指定动画对象*/);
    //主体进行位移，结束时调用onMoveDone方法
    CCAction *action = CCSequence::actions(
        CCMoveBy::actionWithDuration(0.20f, moveByPosition),
        //把方向信息传递给onMoveDone方法
        CCCallFuncND::actionWithTarget(this,
        callfuncND_selector(Hero::onMoveDone),(void*)direction),
        NULL);
    this->runAction(action);
    isHeroMoving = true;
}
```

onMoveDone和setFaceDirection方法基本和原来HelloWorldScene中的实现类似，注意要将用到heroSprite->getPosition()的地方都需要改成this->getPosition()。

接下来开始改写checkCollision方法，大家会发现里面用到了gameMap对象。如何在Hero类中访问GameMap对象呢？我们下一小节会介绍一种方法。

好了，目前位置Hero类的功能已经基本完成，现在需要修改一下GameLayer中初始化勇士的方法：

```
//调用Hero类的静态方法创建实例
hero = Hero::heroWithinLayer();
//设置Hero的起始位置
hero->setPosition(map->positionForTileCoord(ccp(1, 11)));
//将Hero加入GameLayer
addChild(hero);
```

很简单吧，仅需要3行代码就在游戏中创建出了一个勇士。

3. 单例类

大家应该注意到上一小节还有两个小部分没有完成，一是动画相关的代码，二是在Hero类中如何访问GameMap对象。

我们先来完成动画相关的代码。这一部分单独拿出来讲，是考虑到定义动画的工作很耗费代码行数，并且十分单调重复，没有必要放在具体的游戏对象里，可以专门设计一个动画管理器来管理游戏中所有的动画。在场景创建之初，将可能用到的动画，从配置文件一次性加载；在场景运行期间，就可以直接从管理器去得相应的动画资源了。在场景被销毁的时候，再控制动画管理器销毁不用的资源。

好，在实现具体的动画管理器之前，我们需要多方位思考，请先考虑这几个问题：动画管理器究竟会以什么样的形式被调用？在游戏的各个对象中，哪些会有访问动画管理器的需求？不同的类需要访问不同的管理器实例吗？是否会有多线程同时访问？在仔细思考完这些问题以后，相信你已经得出了答案：我们在整个游戏过程中只需要一个动画管理器，它需要以尽可能方便的形式被所有游戏对象访问到，最后的多线程问题在这款游戏中不存在。所以让我们异口同声地说出心中的那两个字吧，没错！就是单例。

我们先写一个单例模式的模板容器，以后就可以方便地实现单例了。模板容器的具体代码如下，其中定义了一个静态变量，用于保存单例的实例；有两个公用方法，用于获取唯一实例和释放操作。

```
#ifndef _SINGLETON_H
#define _SINGLETON_H

template <class T>
class Singleton
{
public:
    //获取类的唯一实例
    static inline T* instance();
    //释放类的唯一实例
    void release();
protected:
    Singleton(void){}
    ~Singleton(void){}
    static T* _instance;
};

template <class T>
inline T* Singleton<T>::instance()
{
    if(!_instance)
        _instance = new T;
    return _instance;
}

template <class T>
void Singleton<T>::release()
{
    if (!_instance)
        return;
```

```
    delete _instance;
    _instance = 0;
}

//cpp文件中需要先声明静态变量
#define DECLARE_SINGLETON_MEMBER(_Ty)\
    template <> _Ty* Singleton<_Ty>::_instance = NULL;

#endif//_SINGLETON_H
```

然后我们新建一个AnimationManager类，继承Singleton类。可以直接重用基类的获取实例和销毁实例的方法。同时，它拥有一个动画映射表，用来缓存动画模板，可以根据int型的键值查询对应的动画模板。定义了一个枚举类型，罗列了目前所有动画对应的键值。对外暴露的公共方法有3个：initAnimationMap用于初始化动画模板资源到AnimationCache中，getAnimation可以根据键值返回对应的动画模板定义，createAnimate方法用来生成指定的动画实例，可以直接供CCNode的runAction调用。具体头文件定义如下：

```
#ifndef _ANIMATION_MANAGER_H_
#define _ANIMATION_MANAGER_H_

#include "MTGame.h"

typedef enum
{
    aDone = 0,//向下行走动画
    aLeft,//向左行走动画
    aRight,//向右行走动画
    aUp,//向上行走动画
} AnimationKey;//动画模板键值

using namespace cocos2d;

class AnimationManager : public Singleton<AnimationManager>
{
public:
    AnimationManager();
    ~AnimationManager();
    //初始化动画模板缓存表
    bool initAnimationMap();
    //根据名字得到一个动画模板
    CCAnimation *getAnimation(int key);
    //创建一个动画实例
    CCAnimate *createAnimate(int key);
protected:
    //加载勇士行走动画模板
    CCAnimation *createHeroMovingAnimationByDirection(HeroDirection direction);
};
//定义动画管理器实例的别名
#define sAnimationMgr AnimationManager::instance()
```

```
#endif
```

各方法的具体实现如下，在initAnimationMap方法中，将勇士的行走动画加载到全局的CCAnimationCache中。getAnimation获取动画模板时，也是从CCAnimationCache中获取。

```
#include "AnimationManager.h"

DECLARE_SINGLETON_MEMBER(AnimationManager);

AnimationManager::AnimationManager()
{
}

AnimationManager::~AnimationManager()
{
    //CCDirector会自己清除AnimationCache
    //CCAnimationCache::purgeSharedAnimationCache();
}

bool AnimationManager::initAnimationMap()
{
        char temp[20];
    sprintf(temp, "%d", aDown);
    //加载勇士向下走的动画
    CCAnimationCache::sharedAnimationCache()->addAnimation(createHeroMovingAnimation-
    ByDirection(kDown), temp);
    sprintf(temp, "%d", aRight);
    //加载勇士向右走的动画
    CCAnimationCache::sharedAnimationCache()->addAnimation(createHeroMovingAnimation-
    ByDirection(kRight), temp);
    sprintf(temp, "%d", aLeft);
    //加载勇士向左走的动画
    CCAnimationCache::sharedAnimationCache()->addAnimation(createHeroMovingAnimation-
    ByDirection(kLeft), temp);
    sprintf(temp, "%d", aUp);
    //加载勇士向上走的动画
    CCAnimationCache::sharedAnimationCache()->addAnimation(createHeroMovingAnimation-
    ByDirection(kUp), temp);

    return true;
}

CCAnimation  *AnimationManager::createHeroMovingAnimationByDirection(HeroDirection-
direction)
{
    CCTexture2D *heroTexture = CCTextureCache::sharedTextureCache()->addImage("hero.png");
    CCSpriteFrame *frame0, *frame1, *frame2, *frame3;
    //第二个参数表示显示区域的x, y, width, height，根据direction来确定显示的y坐标
    frame0 = CCSpriteFrame::frameWithTexture(heroTexture, cocos2d::CCRectMake(32*0,
    32*direction, 32, 32));
    frame1 = CCSpriteFrame::frameWithTexture(heroTexture, cocos2d::CCRectMake(32*1,
    32*direction, 32, 32));
```

```
        frame2 = CCSpriteFrame::frameWithTexture(heroTexture, cocos2d::CCRectMake(32*2,
        32*direction, 32, 32));
        frame3 = CCSpriteFrame::frameWithTexture(heroTexture, cocos2d::CCRectMake(32*3,
        32*direction, 32, 32));
        CCMutableArray<CCSpriteFrame *> *animFrames = new CCMutableArray<CCSpriteFrame*>(4);
        animFrames->addObject(frame0);
        animFrames->addObject(frame1);
        animFrames->addObject(frame2);
        animFrames->addObject(frame3);
        CCAnimation *animation = new CCAnimation();
        //0.05f表示每帧动画间的间隔
        animation->initWithFrames(animFrames, 0.05f);
        animFrames->release();

        return animation;
    }

    //获取指定动画模板
    CCAnimation *AnimationManager::getAnimation(int key)
    {
        char temp[20];
        sprintf(temp, "%d", key);
        return CCAnimationCache::sharedAnimationCache()->animationByName(temp);
    }

    //获取一个指定动画模板的实例
    CCAnimate *AnimationManager::createAnimate(int key)
    {
        //获取指定动画模板
        CCAnimation *anim = getAnimation(key);
        if(anim)
        {
            //根据动画模板生成一个动画实例
            return cocos2d::CCAnimate::actionWithAnimation(anim);
        }
        return NULL;
    }
```

接下来，我们在哪里调用initAnimationMap方法呢？如果游戏资源比较多，最好创建一个加载（loading）界面，在后台加载资源，加载完毕后再进入游戏场景。如果资源不多，可以简单地在AppDelegate的applicationDidFinishLaunching中初始化AnimationManager，然后在AppDelegate的析构函数中释放它。

```
    bool AppDelegate::applicationDidFinishLaunching()
    {
        CCDirector *pDirector = CCDirector::sharedDirector();
        pDirector->setOpenGLView(&CCEGLView::sharedOpenGLView());
        pDirector->setDeviceOrientation(kCCDeviceOrientationLandscapeLeft);
        pDirector->setDisplayFPS(true);
        pDirector->setAnimationInterval(1.0 / 60);
        //初始化动画管理器
        sAnimationMgr->initAnimationMap();
```

```
    //创建游戏主界面
    CCScene *pScene = GameScene::showGameScene();
    //让Director运行场景
    pDirector->runWithScene(pScene);
    return true;
}

AppDelegate::~AppDelegate()
{
    SimpleAudioEngine::end();
    //释放动画管理器
    sAnimationMgr->release();
}
```

在Hero类的move方法中，修改heroSprite的runAction方法如下：

```
heroSprite->runAction(sAnimationMgr->createAnimate(direction));
```

上面我们用单例模式创建了一个动画管理器，在游戏的任意对象中都能方便地使用。接着我们再介绍一种单例模式的应用场景。大家可能已经发现，随着代码的重构，单独的类被拆分成多个功能相对单一的类，这样类之间的数据访问成了一个很大的问题。比如，在Hero类的checkCollision方法中需要用到GameMap对象，在moveDone事件回调中，需要访问GameLayer的setSceneScrollPosition方法。我们当然可以通过传递对象的方式来进行数据交互，但这种方式相对来说比较麻烦。如果可以保证各个对象的唯一性，比如当前游戏主场景、游戏地图和勇士等只可能有一个实例存在，那么最简便的方法就是使用单例模式创建一个类，保存全局都可以访问的唯一性变量。下面，新建一个Global类继承于Singleton类，其中包含了几个需要全局访问的游戏对象，即GameLayer、GameMap和Hero等，具体头文件如下：

```
#ifndef _GLOBAL_H_
#define _GLOBAL_H_

#include "MTGame.h"

using namespace cocos2d;

class GameLayer;
class GameMap;
class Hero;

class Global : public Singleton<Global>
{
public:
    Global(void);
    ~Global(void);

    //游戏主图层
    GameLayer *gameLayer;
    //游戏地图
    GameMap *gameMap;
    //勇士
    Hero *hero;
};
```

```
#define sGlobal Global::instance()
#endif
```

注意，要在各类的构造函数中对Global中的变量进行赋值：

```
Hero::Hero()
{
    sGlobal->hero = this;
}

GameMap::GameMap()
{
    sGlobal->gameMap = this;
}

GameLayer::GameLayer()
{
    sGlobal->gameLayer = this;
}
```

由于我们只保存了上述3个对象的指针，所以对象本身的释放工作不需要Global类来完成，只需要在析构函数中将各个指针设置为NULL即可。

```
Global::~Global(void)
{
    gameLayer = NULL;
    gameMap = NULL;
    hero = NULL;
}
```

创建好了Global类，修改Hero类中所有需要使用gameMap实例的地方，部分代码如下：

```
void Hero::onMoveDone(CCNode *pTarget, void* data)
{
    //将void*先转换为int，再从int转换到枚举类型
    int direction = (int) data;
    setFaceDirection((HeroDirection)direction);
    isHeroMoving = false;
    sGlobal->gameLayer->setSceneScrollPosition(this->getPosition());
}
//判断碰撞类型
CollisionType Hero::checkCollision(CCPoint heroPosition)
{
    //cocos2d-x坐标转换为Tilemap坐标
    CCPoint tileCoord = sGlobal->gameMap->tileCoordForPosition(heroPosition);
    //如果勇士坐标超过地图边界，返回kWall类型，阻止其移动
    if (heroPosition.x < 0 || tileCoord.x > sGlobal->gameMap->getMapSize().width - 1
       || tileCoord.y < 0 || tileCoord.y > sGlobal->gameMap->getMapSize().height - 1)
        return kWall;
    //获取墙壁层对应坐标的图块ID
    int tileGid = sGlobal->gameMap->getWallLayer()->tileGIDAt(tileCoord);
```

```
    //如果图块ID不为0，表示有墙
    if (tileGid) {
        return kWall;
    }
    //可以通行
    return kNone;
}
```

4. 控制层

前面一节，我们抽象出了Hero类，增加了Singleton容器类，并用其创建了动画管理器AnimationManager和一个存放全局变量的类Global。目前的GameLayer类看起来清爽了很多，地图和勇士的初始化和相关方法都移到相关的类中，仅剩余了下面几个部分。

(1) setSceneScrollPosition方法，用于根据勇士位置计算场景滚动的距离。

(2) 一个定时器方法，用于每帧调用setSceneScrollPosition。

(3) 关闭按钮及其对应的onClick方法。

(4) 4个方向按钮及onClick方法。

大家应该已经看出，后面两项也可以从GameLayer中分离出来，因为它们并不属于游戏内容相关的对象，可以把它们归结为控制层面的一部分。因此，我们最好创建一个控制层来统一管理相关的按钮。控制层应该作为一个独立的CCLayer，它的zOrder需要比GameLayer更高。首先创建一个空的类，即ControlLayer，它应该包括CCLayer标准的初始化方法和按钮点击事件的回调函数，ControlLayer 的头文件代码如下：

```
#ifndef __CONTROLLAYER_H__
#define __CONTROLLAYER_H__

#include "MTGame.h"

using namespace cocos2d;

class ControlLayer: public CCLayer
{
public:
    ControlLayer(void);
    ~ControlLayer(void);
    //node方法会调用此函数
    virtual bool init();
    //方向按钮点击事件的回调
    void menuCallBackMove(CCObject* pSender);
    //关闭按钮点击事件的回调
    void menuCloseCallback(CCObject* pSender);
    //使用CCLayer标准的创建实例的方式，声明node方法
    LAYER_NODE_FUNC(ControlLayer);
};

#endif
```

ControlLayer 的代码文件中的具体实现只需要将GameLayer中的相关代码复制过来即可：

```
#include "MTGame.h"
ControlLayer::ControlLayer(void)
{
}

ControlLayer::~ControlLayer(void)
{
}

bool ControlLayer::init()
{
    if ( !CCLayer::init() )
    {
        return false;
    }
    //创建关闭按钮
    CCMenuItemImage *pCloseItem = CCMenuItemImage::itemFromNormalImage(
        "CloseNormal.png",
        "CloseSelected.png",
        this,
        menu_selector(ControlLayer::menuCloseCallback));
    pCloseItem->setPosition(ccp(CCDirector::sharedDirector()->getWinSize().width
    - 20, 20));
    CCMenu *pMenu = CCMenu::menuWithItems(pCloseItem, NULL);
    pMenu->setPosition(CCPointZero);
    this->addChild(pMenu, 1);
    //创建方向按钮
    CCMenuItem  *down  =  CCMenuItemFont::itemFromString("down",  this,  menu_selector
    (ControlLayer::menuCallBackMove));
    CCMenuItem  *left  =  CCMenuItemFont::itemFromString("left",  this,  menu_selector
    (ControlLayer::menuCallBackMove) );
    CCMenuItem *right = CCMenuItemFont::itemFromString("right", this, menu_selector
    (ControlLayer::menuCallBackMove) );
    CCMenuItem *up = CCMenuItemFont::itemFromString("up", this, menu_selector
    (ControlLayer::menuCallBackMove) );
    CCMenu *menu = CCMenu::menuWithItems(down, left, right, up, NULL);
    //为了方便查找，给每个menuItem设置Tag
    down->setTag(kDown);
    left->setTag(kLeft);
    right->setTag(kRight);
    up->setTag(kUp);
    //菜单项按间距水平排列
    menu->alignItemsHorizontallyWithPadding(50);
    addChild(menu);
    return true;
}

void ControlLayer::menuCloseCallback(CCObject *pSender)
{
    CCDirector::sharedDirector()->end();
}

void ControlLayer::menuCallBackMove(CCObject *pSender)
```

```
{
     CCNode *node = (CCNode *) pSender;
    //按钮的Tag就是需要行走的方向
    int targetDirection = node->getTag();
    sGlobal->hero->move((HeroDirection) targetDirection);
}
```

最后，我们修改GameScene的init方法，在里面新建一个ControlLayer实例，并把它加为自己的孩子节点：

```
//新建一个ControlLayer实例
ControlLayer *controlLayer = ControlLayer::node();
CC_BREAK_IF(! controlLayer);
//将ControlLayer实例添加到场景中
scene->addChild(controlLayer);
```

11.1.3 小结

通过不懈的努力，目前代码看起来已经比较整洁了，各个类分工明确。当然，我们还有一些优化的工作可以做，下面给出两个。

(1) 将用到的常量放到一个公用的头文件GameConstants.h中。然后将GameConstants.h包含到MTGame.h中，这样所有游戏对象都可以访问这些常量了。

```
#ifndef __GAME_CONSTANTS_H__
#define __GAME_CONSTANTS_H__

typedef enum {
    kDown =  0,//向下方向
    kLeft = 1,//向左方向
    kRight= 2,//向右方向
    kUp = 3,//向上方向
    kNormal,
} HeroDirection;//勇士方向

typedef enum
{
    kNone = 1,//可以通行
    kWall,//墙
    kEnemy,//敌人
} CollisionType;//碰撞类型

typedef enum
{
    aDone = 0,//向下行走动画
    aLeft,//向左行走动画
    aRight,//向右行走动画
    aUp,//向上行走动画
} AnimationKey;//动画模板键值

#endif
```

(2) 创建不同的文件夹，将类再继续分类。我的归类方式如图11-1所示。

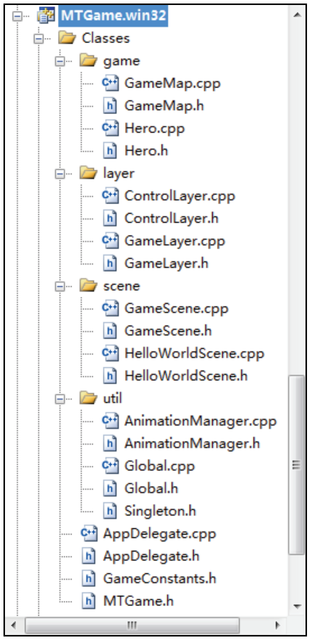


图11-1　将类归类到不同文件夹下

将类移动到不同文件夹后，还需要修改MTGame.h中对应头文件的路径，注意Singleton.h要在AnimationManager.h和Global.h之前声明，防止编译时找不到基类：

```
#ifndef __MTGAME_H__
#define __MTGAME_H__

#include "cocos2d.h"
#include "GameConstants.h"

#include "layer/ControlLayer.h"
#include "layer/GameLayer.h"

#include "scene/GameScene.h"

#include "game/GameMap.h"
#include "game/Hero.h"

#include "util/Singleton.h"
#include "util/AnimationManager.h"
#include "util/Global.h"

#endif
```

现在对代码的重构工作终于告一段落了，让我们总结一下前面的工作。

(1) 首先将HelloWorldScene类拆分为GameScene和GameLayer。分离出了功能相对单一的GameScene类，用于创建游戏场景。

(2) 然后我们从GameLayer类中分离出了GameMap类。将游戏地图的创建、Tilemap和cocos2d-x坐标系的转换以及碰撞检测方法包装在了GameMap类中。

(3) 接下来，创建了一个Hero类，继承于CCNode，其中包含了一个用于显示勇士主体的CCSprite，未来可以包含更多依附于勇士主体的精灵。同时，Hero类提供了静态的创建实例的方法和向各个方向移动的方法。

(4) 随后，写了一个模板容器类Singleton用来实现单例模式，并以此制作了动画管理器。达到了让各个游戏对象可以方便创建动画的目的。又根据Singleton类创建了Global类，方便游戏对象之间的相互访问，并且把一些常量提取到GameConstants文件中。

(5) 最后，将控制按钮从GameLayer提取出来，放到了控制层ControlLayer中。在GameScene中将ControlLayer放置于GameLayer上方。

这一节我们虽然没有添加任何新功能，但是还是很有意义，因为一方面可以进一步消化以前的代码，另一方面还对以后游戏功能的扩展奠定了坚实的基础。

11.2 添加更多游戏元素

目前我们的勇士只能在空旷的地图上来回乱转，既不能与怪物进行战斗，也无法拾取物品，也不能和NPC对话等。这一节的目的就是向游戏内添加更多的元素，丰富游戏内容，增加可玩性，如图11-2所示。



图11-2 为游戏添加更多元素

11.2.1　添加怪物

魔塔的地图上充满了强弱不一的怪物，它们都是勇士的敌人。勇士必须打败它们才能不断地探索地图的未知区域。这一节，我们将依次讨论以下内容：如何在TiledMap上绘制怪物，如何让怪物动起来，如何让勇士遇到怪物时可以战斗，以及如何设定怪物的属性等。

1. 在地图上绘制怪物

打开Tiled编辑器，载入我们一开始绘制的0.tmx地图。在图层面板中选择添加块层，创建完毕后重命名为enemy，置于wall层的上方。将怪物动画素材enemy.png（图11-3所示的是素材的部分截图）复制到工程的Resource目录下。



图11-3　怪物动画素材

选择菜单栏中的“地图→新图块”，导入enemy.png。现在就可以使用新导入的素材在enemy图层上绘制怪物了，请充分发挥你的想象力吧。图11-4所示的是我在地图上添加完怪物后的效果。

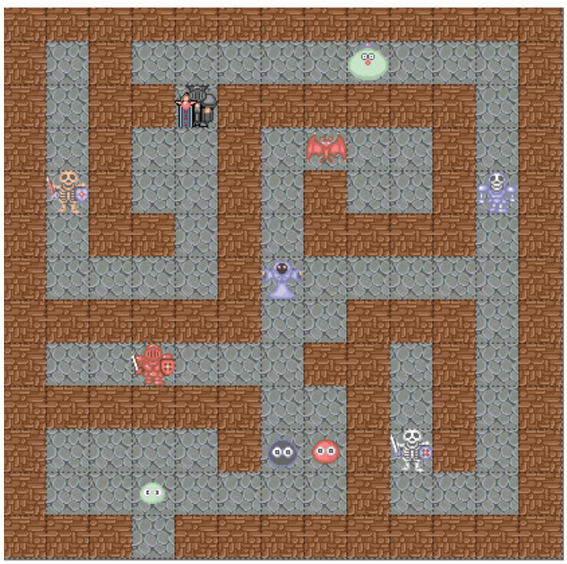


图11-4　在游戏地图上添加怪物

让我们先运行下游戏，看看效果，发现怪物已经被正确地绘制到地图上了，如图11-5所示。但是让人郁闷的是，怪物都是静止不动的，好像纸片一样贴在地图上。下面我们就想办法让地图上的图块动起来。



图11-5　添加怪物后的游戏运行界面

2. 添加怪物动画

怪物在地图上的位置是不变的，但它们都有对应的原地站立时的动画。大家可能想到使用前面创建的动画管理器来创建动画模板，然后播放各个怪物精灵的动画。这样做没错，但是想想就觉得麻烦：如果有100种怪物的话，难道要定义100个动画模板吗？答案是否定的。那么该如何实现怪物原地站立的动画呢？从提供的怪物图片素材可以发现：每个怪物的动作由4帧动画组成，且每帧尺寸一致，都正好是一个图块的大小。是否可以定时地更新怪物对应的图块，从而产生动画的效果？结论是可以。cocos2d-x工程下tests项目中的TMXReadWriteTest例子演示了如何动态改变TileMap上的图块，以及用定时器不停地重复更新图块的工作。关键代码如下：

```
//创建定时器，反复更新图块
schedule(schedule_selector(TMXReadWriteTest::repaintWithGID), 2.0f);

void TMXReadWriteTest::repaintWithGID(ccTime dt)
{
      //获TileMap地图对象
    CCTMXTiledMap *map = (CCTMXTiledMap*)getChildByTag(kTagTileMap);
      //获取第0层
    CCTMXLayer *layer = (CCTMXLayer*)map->getChildByTag(0);
    CCSize s = layer->getLayerSize();
      //遍历一行
```

```
    for( int x=0; x<s.width;x++)
    {
        //倒数第二行
        int y = (int)s.height-1;
        //获取指定位置当前的图块ID
        unsigned int tmpgid = layer->tileGIDAt( ccp((float)x, (float)y) );
        //更新指定位置的图块ID
        layer->setTileGID(tmpgid+1, ccp((float)x, (float)y));
    }
}
```

明白了上面的代码我们就对实现怪物动画有清晰的思路了：创建一个定时器，在定时器中遍历所有怪物，令其图块ID加一，如果动画完成，则将图块ID置回初始的数值。

下面请大家思考一下如何遍历所有怪物呢？由于遍历方法需要在定时器中调用，就必须尽可能优化其速度。tests中的例子使用了二维数组的遍历。但设想一下，如果地图很大，遍历起来会很耗时。另外，怪物的分布是稀疏的，二维数组中的大部分元素都有可能为空。所以直接在定时器中遍历地图上的每个格子是不明智的。

那么我们可以在地图初始化完毕后，先做一个预处理：将每个怪物的方位，初始时的图块ID存放到一个数组中。然后在定时器中遍历这个数组就可以了。

首先创建一个Enemy类，用于存放怪物的方位和初始的图块ID，将来还会存放怪物的属性等信息：

```
#ifndef _ENEMY_H_
#define _ENEMY_H_

#include "cocos2d.h"

using namespace cocos2d;

class Enemy : public CCObject
{
public:
    Enemy(void);
    ~Enemy(void);
    //怪物在TileMap上的方位
    CCPoint position;
    //怪物初始的图块ID
    int startGID;
};

#endif
```

然后修改GameMap的初始化方法，遍历TileMap上的所有怪物，生成Enemy对象，添加到一个CCMutabalArray中。随后启动一个定时器，每隔0.2 s更新一次怪物动画。具体代码如下：

```
//TliedMap额外的初始化方法
void GameMap::extraInit()
{
```

```
    //开启各个图层的纹理抗锯齿
    enableAnitiAliasForEachLayer();
    //初始化地板层和墙壁层对象
    floorLayer = this->layerNamed("floor");
    wallLayer = this->layerNamed("wall");
    //获取怪物层
    enemyLayer = this->layerNamed("enemy");
    CCSize s = enemyLayer->getLayerSize();
    enemyArray = new CCMutableArray<Enemy*>();
    //遍历enemy层，将存在的怪物保存到数组中
    for (int x = 0; x < s.width; x++) {
        for (int y = 0; y < s.height; y++) {
            int gid = enemyLayer->tileGIDAt(ccp(x, y));
            if (gid != 0) {
                Enemy *enemy = new Enemy();
                //保存怪物坐标
                enemy->position = ccp(x, y);
                //保存怪物起始的图块ID
                enemy->startGID = gid;
                //添加怪物对象到数组
                enemyArray->addObject(enemy);
            }
        }
    }
    //用于更新敌人动画
    schedule(schedule_selector(GameMap::updateEnemyAnimation), 0.2f);
}
```

在updateEnemyAnimation中，需要遍历enemyArray，并计算动画下一帧对应的图块ID，具体代码如下：

```
//更新怪物的图块
void GameMap::updateEnemyAnimation(ccTime dt)
{
    CCMutableArray<Enemy*>::CCMutableArrayIterator  iter;
    Enemy *enemy;
    //遍历保存所有怪物对象的数组
    for (iter = enemyArray->begin(); iter != enemyArray->end(); ++iter) {
        enemy = *iter;
        if (enemy != NULL) {
            //获取怪物当前的图块ID
            int gid = enemyLayer->tileGIDAt(enemy->position);
            gid++;
            //如果下一个图块ID 起始图块ID
            if (gid - enemy->startGID > 3) {
                gid = enemy->startGID;
            }
            //给怪物设置新的图块
            enemyLayer->setTileGID(gid, enemy->position);
        } else {
            break;
```

```
            }
        }
    }
```

现在编译运行代码，发现地图上的怪物们都动起来了。注意，在GameMap的析构方法里要释放enemyArray和定时器：

```
GameMap::~GameMap(void)
{
    this->unscheduleAllSelectors();
    enemyArray->release();
}
```

3. 与怪物进行战斗

这一节的目标是实现勇士与怪物战斗的效果。我们希望勇士在遭遇敌人时，可以显示战斗动画，并且怪物身上显示被打击的效果，勇士头上冒出一行文字提示损失的生命值；战斗结束后，怪物从地图上消失。

首先我们准备好了一个战斗动画效果：一个舞动的刀光，如图11-6所示。

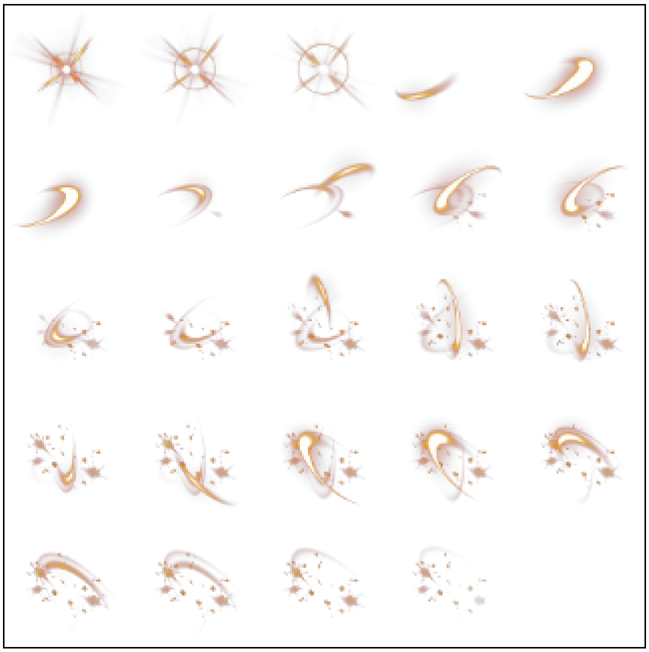


图11-6 舞动的刀光素材

它实际起作用的是4, 6, 8, 10, 13, 15, 17, 19, 20, 22帧，并且第17帧和第19帧在y方向上有−8的偏移量。在GameConstants.h中新增一个动画模板键值aFight，然后在动画管理器中新增一个方

法createFightAnimation，用于创建战斗动画模板：

```
//创建战斗动画模板
CCAnimation *AnimationManager::createFightAnimation()
{
    //定义每帧的序号
    int fightAnim[] =
    {
        4,6,8,10,13,15,17,19,20,22
    };
    CCMutableArray<CCSpriteFrame *> *animFrames = new CCMutableArray<CCSpriteFrame*>();
    CCTexture2D *texture = CCTextureCache::sharedTextureCache()->addImage("sword.png");
    CCSpriteFrame *frame;
    int x, y;
    for (int i = 0; i < 10; i++)
    {
        //计算每帧在整个纹理中的偏移量
        x = fightAnim[i] % 5 - 1;
        y = fightAnim[i] / 5;
        frame = CCSpriteFrame::frameWithTexture(texture, cocos2d::CCRectMake(192*x,
        192*y, 192, 192));
        //第17帧和第19帧在y方向上有-8的偏移量
        if (fightAnim[i] == 17 || fightAnim[i] == 19)
        {
            frame->setOffsetInPixels( ccp(0, -8) );
        }
        animFrames->addObject(frame);
    }
    CCAnimation *animation = new CCAnimation();
    animation->initWithFrames(animFrames, 0.1f);
    animFrames->release();
    return animation;
}
```

先把准备好的刀光动画放在一边，下面我们来实现勇士遇到怪物时的检测碰撞。方法与检测墙壁碰撞基本一致：将勇士移动的目标位置由cocos2d-x坐标系转换为TileMap坐标系，通过CCTMXLayer的tileGIDAt方法获得图块ID，判断碰撞类型。修改GameMap的checkCollision方法如下：

```
//判断碰撞类型
CollisionType Hero::checkCollision(CCPoint heroPosition)
{
    //cocos2d-x坐标转换为Tilemap坐标
    targetTileCoord = sGlobal->gameMap->tileCoordForPosition(heroPosition);
    //如果勇士坐标超过地图边界，返回kWall类型，阻止其移动
    if (heroPosition.x < 0 || targetTileCoord.x > sGlobal->gameMap->getMapSize().width
        - 1 || targetTileCoord.y < 0 || targetTileCoord.y > sGlobal->gameMap->getMap-
        Size() .height - 1)
        return kWall;
    //获取墙壁层对应坐标的图块ID
```

```
    int tileGid = sGlobal->gameMap->getWallLayer()->tileGIDAt(targetTileCoord);
    //如果图块ID不为0，表示有墙
    if (tileGid) {
        return kWall;
    }
    //获取怪物层对应坐标的图块ID
    tileGid = sGlobal->gameMap->getEnemyLayer()->tileGIDAt(targetTileCoord);
    //如果图块ID不为0，表示有怪物
    if (tileGid) {
        //开始战斗
        fight();
        return kEnemy;
    }
    //可以通行
    return kNone;
}
```

其中fight方法封装了勇士和怪物开始战斗后的逻辑，包括显示怪物受打击的效果，播放战斗动画，显示损失的生命值等。

第一步先来实现怪物受打击效果——每隔0.4 s显示一次红色的状态，共重复3次。我们在GameMap中新建一个方法showEnemyHitEffect，根据入参怪物在TileMap上所在的坐标取得对应位置的CCSprite对象，调用一个间隔为0.2 s的定时器，将CCSprite对象的颜色在白色和红色之间切换，反复切换5次后取消定时器。具体代码如下：

```
//显示怪物打击动画
void GameMap::showEnemyHitEffect(CCPoint tileCoord)
{
    //更新次数
    fightCount = 0;
    //获取怪物对应的CCSprite对象
    fightingEnemy = enemyLayer->tileAt(tileCoord);
    if (fightingEnemy == NULL)
        return;
    //设置怪物sprite颜色为红色
    fightingEnemy->setColor(ccRED);
    //启动定时器更新打击状态
    this->schedule(schedule_selector(GameMap::updateEnemyHitEffect), 0.18f);
}
//更新怪物战斗时的颜色状态
void GameMap::updateEnemyHitEffect(ccTime dt)
{
    //更新次数加一
    fightCount++;
    if (fightCount % 2 == 1) {
        //设置怪物精灵颜色为白色
        fightingEnemy->setColor(ccWHITE);
    } else {
        //设置怪物精灵颜色为红色
        fightingEnemy->setColor(ccRED);
```

```
    }
    //切换5次后取消定时器
    if (fightCount == 5)
    {
        unschedule(schedule_selector(GameMap::updateEnemyHitEffect));
    }
}
```

第二步我们希望在战斗的同时，勇士头上会飘出一行文字，即诸如“生命值：−100”类的提示。在GameLayer中实现一个公有方法showTip，创建一个CCLabelTTF对象，让它执行向上移动进入、停顿、淡出的动画序列。在动画结束的回调函数中，将CCLabelTTF对象删除。

```
//显示提示信息
void GameLayer::showTip(const char *tip, CCPoint startPosition)
{
    //新建一个文本标签
    CCLabelTTF *tipLabel = CCLabelTTF::labelWithString(tip, "Arial", 20);
    tipLabel->setPosition(ccpAdd(startPosition, ccp(16, 16)));
    this->addChild(tipLabel, kZTip,kZTip);
    //定义动画效果
    CCAction *action = CCSequence::actions(
        CCMoveBy::actionWithDuration(0.5f, ccp(0, 32)),
        CCDelayTime::actionWithDuration(0.5f),
        CCFadeOut::actionWithDuration(0.2f),
        CCCallFuncN::actionWithTarget(this, callfuncN_selector(GameLayer::onShowTipDone)),
        NULL);
    tipLabel->runAction(action);
}
//提示消息显示完后的回调
void GameLayer::onShowTipDone(CCNode* pSender)
{
    //删掉文本标签
    this->getChildByTag(kZTip)->removeFromParentAndCleanup(true);
}
```

第三步我们来实现战斗时的动画效果，这时之前准备好的刀光动画就可以派上用场了。在heroInit方法中定义一个空的CCSprite用于显示战斗动画，在fight方法中，让该精灵执行定义好的刀光动画，在战斗结束的回调函数中，删除怪物对应位置的图块。另外，显示怪物打击效果以及损失生命值的方法也在下面列出。注意，方法的开头会判断勇士是否已经在战斗状态，如果是，则不响应新的战斗请求。

```
//开始战斗
void Hero::fight()
{
    //已经在战斗中，避免重复战斗
    if (isHeroFighting)
        return;
    isHeroFighting = true;
    //显示怪物受到打击的效果
    sGlobal->gameMap->showEnemyHitEffect(targetTileCoord);
```

```
    //显示损失的生命值，先用假数据替代一下
    char temp[30] = {0};
    sprintf(temp, "lost hp: -%d", 100);
    sGlobal->gameLayer->showTip(temp, getPosition());
    //将用于显示战斗动画的精灵设置为可见
    fightSprite->setIsVisible(true);
    //计算显示战斗动画的位置为勇士和怪物的中间点
    CCPoint pos = ccp((targetPosition.x - getPosition().x) / 2 + 16, (targetPosition.y
    - getPosition().y) / 2 + 16);
    fightSprite->setPosition(pos);
    //创建战斗动画
    CCAction *action = CCSequence::actions(
        sAnimationMgr->createAnimate(aFight),
        CCCallFuncN::actionWithTarget(this,callfuncN_selector(Hero::onFightDone)),
        NULL);
    fightSprite->runAction(action);
}
//战斗结束的回调
void Hero::onFightDone(CCNode *pSender)
{
    //删除怪物对应的图块，表示已经被消灭
    sGlobal->gameMap->getEnemyLayer()->removeTileAt(targetTileCoord);
    isHeroFighting = false;
}
```

注意，在战斗结束后，除了从TileMap地图上删除图块以外，还需要更新GameMap中保存怪物对象的数组enemyArray。最好不要直接删除，因为有一个定时器在不停地更新怪物动画，正确的方法是在updateEnemyAnimation中判断是否有怪物的图块ID为0，有则说明已经被删除了，记录下此元素，在循环外再进行删除操作。修改updateEnemyAnimation方法如下：

```
//更新怪物的图块
void GameMap::updateEnemyAnimation(ccTime dt)
{
    CCMutableArray<Enemy*>::CCMutableArrayIterator  iter;
    Enemy *enemy, *needRemove = NULL;
    //遍历保存所有怪物对象的数组
    for (iter = enemyArray->begin(); iter != enemyArray->end(); ++iter) {
        enemy = *iter;
        if (enemy != NULL) {
                //获取怪物当前的图块ID
                int gid = enemyLayer->tileGIDAt(enemy->position);
                //如果怪物被删除了，需要把它在enemyArray中也删除
                if (gid == 0)
                {
                    needRemove = enemy;
                    continue;
                    }
                gid++;
                    //如果下一个图块ID 起始图块ID
                if (gid - enemy->startGID > 3) {
                     gid = enemy->startGID;
                }
```

```
            //给怪物设置新的图块
            enemyLayer->setTileGID(gid, enemy->position);
        }
    }
    //删除被消灭的怪物
    if (needRemove != NULL) {
        enemyArray->removeObject(needRemove, true);
    }
}
```

现在可以很高兴地通知大家，这一节的目标已经基本达成！编译运行程序，现在我们的小勇士终于可以与怪物战斗了。

11.2.2　添加物品和门

上一节中，我们在TileMap地图上添加了怪物，并且实现了勇士与怪物战斗的效果。这一节会继续向游戏地图中添加更多的元素，包括各种物品和门。物品有诸如血瓶、钥匙、各种道具和装备之类，勇士可以拾取物品，并获得相应的属性提升或技能，而门只有3种：黄门、蓝门和红门，分别用对应颜色的钥匙才能打开。

1. 在地图上绘制物品和门

切换到Tiled编辑器，打开游戏地图文件0.tmx。首先添加两个块层item和door，分别用于绘制物品和门。将素材item.png（图11-7）和door.png（图11-8）复制到工程的Resource目录下。



图11-7　部分物品的素材

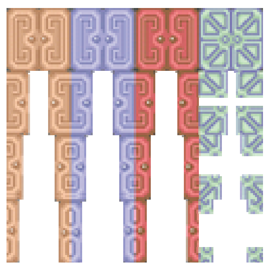


图11-8　门的素材

选择菜单栏中的“地图→新图块”，导入两个图片素材，然后就可以绘制物品和门了。图11-9所示的是我画的地图。

接下来直接运行游戏，我们发现地图现在充实多了，有点真正游戏的样子了，如图11-10所示。

2. 添加物品

我们希望在勇士遇到地图上的物品时，头上会冒出一行文字说明，如“获取小血瓶，生命+100”，然后将地图上的物品移除，并增加到勇士自己的背包当中。下面来看代码实现。

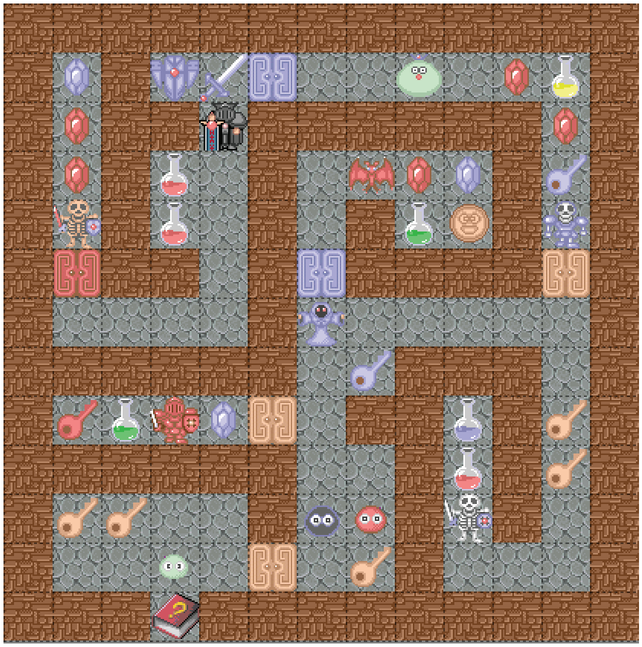


图11-9 在游戏地图上添加物品和门



图11-10 添加物品和门后的游戏运行界面

首先要在GameConstants.h中增加一种碰撞类型`kItem`，接着为GameMap类增加获取`itemLayer`的方法：

```
CC_PROPERTY_READONLY(CCTMXLayer*, itemLayer, ItemLayer);
```

记得要在extraInit方法中为itemLayer和doorLayer赋值：

```
itemLayer = this->layerNamed("item");
```

然后在Hero的checkCollision方法中添加itemLayer的碰撞检测，方法与墙壁和怪物的碰撞检测一致：

```
//获取物品层对应坐标的图块ID
tileGid = sGlobal->gameMap->getItemLayer()->tileGIDAt(targetTileCoord);
//如果图块ID不为0，表示有物品
if (tileGid) {
    //拾取物品
    pickUpItem();
    return kItem;
}
```

目前我们没有对物品进行区分。未来的拾取物品逻辑可以在这个方法中补充，根据物品的ID在地图上显示不同的提示语，以及获得不同的功效等。pickUpItem方法的实现如下：

```
//拾取物品
void Hero::pickUpItem()
{
    //显示提示消息
    sGlobal->gameLayer->showTip("get an item, hp +100", this->getPosition());
    //将物品从地图上移除
    sGlobal->gameMap->getItemLayer()->removeTileAt(targetTileCoord);
}
```

我们还需要对Hero的move方法稍作修改，目前在checkCollision返回不为kNone类型时，就不允许勇士移动。但拾取物品时，勇士仍然可以行走，所以需要修改判断是否允许移动的条件为kWall和kEnemy时禁止移动，而kNone和kItem时允许勇士通行：

```
//调用checkCollision检测碰撞类型，如果是墙壁或怪物，则只需要设置勇士的朝向
CollisionType collisionType = checkCollision(targetPosition);
if (collisionType == kWall || collisionType == kEnemy)
{
    setFaceDirection((HeroDirection)direction);
    return;
}
```

3. 添加门

在地图上添加门的方法仍然与添加墙壁、怪物和物品的方法基本相同。不同之处是，勇士在执行打开门的操作后，门需要播放一段打开的动画。

还是一步一步来，在GameConstants.h中增加一种碰撞类型kDoor。在GameMap中增加doorLayer的相关代码，包括初始化和get方法。这里就不再列出了。接下来修改Hero的checkCollision方法，增加门的碰撞检测：

```
//获取门层对应坐标的图块ID
targetTileGID = sGlobal->gameMap->getDoorLayer()->tileGIDAt(targetTileCoord);
//如果图块ID不为0，表示有门
if (targetTileGID) {
    //打开门
    openDoor(targetTileGID);
    return kDoor;
}
```

openDoor方法的实现如下，其中我们需要保存正在打开的门的图块ID，在更新动画时需要用到。然后开启一个定时器每隔0.1 s更新门的动画。

```
//打开门
void Hero::openDoor(int gid)
{
    //如果门正在被开启，则返回
    if (isDoorOpening)
        return;
    //保存正在被开启的门的初始GID
    targetDoorGID = gid;
    isDoorOpening = true;
    //定时器更新门动画
    schedule(schedule_selector(Hero::updateOpenDoorAnimation), 0.1f);
}
```

在updateOpenDoorAnimation方法中，先计算下一帧的图块ID。我们可以根据图10-24中每帧的位置来计算：TileMap的图块编号方式是横向递增1。拿黄门举例，第1行第1列为第1帧的图块，假设其ID为startGID，那么第2行相应位置的图块ID就为startGID+4，第3帧和第4帧的图块ID分别为startGID+8和startGID+12。计算出下一帧的图块ID后，判断其是否超过startGID+12，是则说明动画结束，可以将门从地图上删除，并且取消定时器，否则更新动画至下一帧。

```
//更新开门动画
void Hero::updateOpenDoorAnimation(ccTime dt)
{
    //计算动画下一帧的图块ID，TileMap的图块编号方式是横向递增1，所以每列相同的位置的图块ID相差了
    //每行图块的个数
    int nextGID = sGlobal->gameMap->getDoorLayer()->tileGIDAt(targetTileCoord) + 4;
    //如果超过了第4帧动画，就将当前位置的图块删除，并取消定时器
    if (nextGID - targetDoorGID > 4 * 3) {
        sGlobal->gameMap->getDoorLayer()->removeTileAt(targetTileCoord);
        unschedule(schedule_selector(Hero::updateOpenDoorAnimation));
        isDoorOpening = false;
    } else {
        //更新动画至下一帧
        sGlobal->gameMap->getDoorLayer()->setTileGID(nextGID, targetTileCoord);
    }
}
```

最后，还要修改一下Hero的move方法，将遇到门的情形归结到禁止勇士移动的种类中：

```
//调用checkCollision检测碰撞类型，如果是墙壁、怪物、门，则只需要设置勇士的朝向
CollisionType collisionType = checkCollision(targetPosition);
if (collisionType == kWall || collisionType == kEnemy || collisionType == kDoor)
{
    setFaceDirection((HeroDirection)direction);
    return;
}
```

到目前为止，我们成功地在游戏地图中添加了物品和门，并且实现了勇士和它们之间的交互。相信大家都已经熟悉了在TileMap上添加元素的方法。那么就请尽情地发挥想象力，为游戏扩展更丰富多彩的功能吧。

11.2.3　添加对象层

前面两节我们在TiledMap上创建图层的时候，选择的都是“添加块层”。细心的同学可能发现还有一个“添加对象层”的选项。对象层有什么作用呢？我们前面创建的怪物、物品和门，相同元素在游戏中的行为都是一致的（相同的怪物属性一致，相同的物品功能一致），所以用重复的图块来表示比较方便。但是RPG游戏中包含的元素不会是千篇一律的，很可能一个游戏对象的行为是独有的。最典型的例子就是RPG中的NPC（Non-Player-Controlled Character），每个NPC都会有自己特有的剧情。这种情形下，用图块来定义这些对象的行为就不合适了。TileMap对象层的元素可以方便地配置属性，拥有自己的*x*、*y*坐标及尺寸大小，非常适合定义有独特行为的游戏元素。除了上面提到的NPC，还有魔塔各层之间的传送门和勇士的出生地点等都比较适合用对象来表示。

1. 添加NPC对象

下面通过添加一个NPC来演示如何使用TileMap的对象层。我们准备在游戏地图中放上一个仙子，可以和她进行对话。图11-11所示是仙子的图片素材，由4帧的动画组成，将其复制到Resource目录下，命名为npc.png。

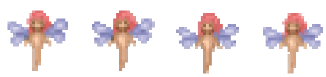


图11-11　仙子的图片素材

打开Tiled编辑器，在图层面板中添加一个对象层，重命名为object。选中object图层，在工具栏中选择插入对象按钮，在地图的(4, 10)处放置一个对象，宽和高皆为1。选中视图菜单中的对齐网格选项，可以在绘制对象的时候自动对齐。放置完毕后效果如图11-12所示，地图左下角的灰色方框就是NPC对象：

接下来，选择工具栏中的“选择对象”按钮（见图11-13），在NPC对象上点击鼠标右键，选择对象属性就会弹出一个对话框，可以在里面配置对象属性。请按照图11-14填写各项信息，除了对象的名称和类型，我们还自定义了3个键值对：`image`表示该对象对应的图片纹理，`rectX`表示在纹理中的起始*x*坐标，`rectY`代表在纹理中的起始*y*坐标。



图11-12 在地图上放置NPC对象



图11-13 选择对象按钮

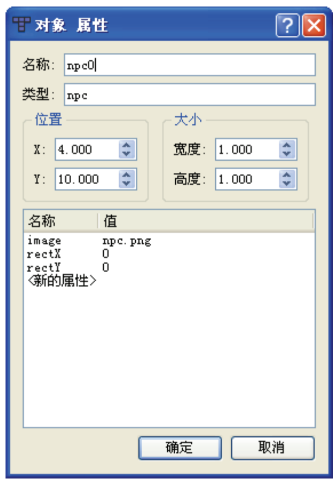


图11-14 NPC对象的属性配置界面

到这里，地图的绘制工作基本完成了，保存文件，开始编写代码。首先，我们希望用一个NPC类来保存上述配置的信息，包括名称、类型和纹理信息等。NPC类的头文件如下：

```
#ifndef __NPC_H__
#define __NPC_H__

#include "MTGame.h"
```

```
using namespace cocos2d;

class NPC : public CCObject
{
public:
    //构造函数中要根据传递的属性表初始化各个变量
    NPC(CCStringToStringDictionary *dict, int x, int y);
    ~NPC(void);
    //用于显示NPC的精灵
    CCSprite *npcSprite;
    //保存在TileMap中配置的name项
    CCString *npcId;
    //NPC所在的TileMap坐标
    CCPoint position;
    //图片纹理的文件路径
    CCString *imagePath;
    //纹理的rect
    CCRect rect;
    //对应配置中的type项
    CCString *type;
};

#endif
```

NPC类的具体实现方法如下：

```
#include "NPC.h"

NPC::NPC(CCStringToStringDictionary *dict, int x, int y)
{
    //获取名称
    std::string key = "name";
    npcId = dict->objectForKey(key);
    //获取类型
    key = "type";
    type = dict->objectForKey(key);
    //获取image项
    key = "image";
    imagePath = dict->objectForKey(key);
    //获取rectX和rectY
    key = "rectX";
    int x1 = dict->objectForKey(key)->toInt();
    key = "rectY";
    int y1 = dict->objectForKey(key)->toInt();
    rect = CCRectMake(x1, y1, 32, 32);
    //position为cocos2d-x坐标，tileCoord为TileMap坐标
    CCPoint position = ccp(x, y);
    tileCoord = sGlobal->gameMap->tileCoordForPosition(position);
    //创建用于显示NPC的精灵
    npcSprite = CCSprite::spriteWithFile(imagePath->m_sString.c_str(), rect);
    npcSprite->setAnchorPoint(CCPointZero);
    npcSprite->setPosition(position);
    sGlobal->gameLayer->addChild(npcSprite, kZNPC);
    //从动画管理器中根据npcId获取动画，开始永久播放
```

```
    CCAnimate *animation = sAnimationMgr->createAnimate(npcId->m_sString.c_str());
    if (animation != NULL) {
        CCActionInterval *action = CCRepeatForever::actionWithAction(animation);
        npcSprite->runAction(action);
    }
}

NPC::~NPC(void)
{
    //释放CCString对象
    CC_SAFE_RELEASE(npcId)
    CC_SAFE_RELEASE(imagePath)
    CC_SAFE_RELEASE(type)
}
```

在构造函数中，我们创建了一个CCSprite用于在地图上显示NPC，然后从Animation-Manager中根据npcId取得对应的动画，让CCSprite不停地执行。动画对应的图片素材为图10-27所示。在AnimationManager的初始化方法中，加载此动画，具体代码如下：

```
//加载NPC动画
CCAnimationCache::sharedAnimationCache()->addAnimation(createNPCAnimation(), "npc0");
```

createNPCAnimation方法的具体实现如下：

```
CCAnimation *AnimationManager::createNPCAnimation()
{
    CCTexture2D *heroTexture = CCTextureCache::sharedTextureCache()->addImage("npc.png");
    CCSpriteFrame *frame0, *frame1, *frame2, *frame3;
    //第二个参数表示显示区域的x, y, width, height
  frame0 = CCSpriteFrame::frameWithTexture(heroTexture, cocos2d::CCRectMake(32*0, 0,
  32, 32));
  frame1 = CCSpriteFrame::frameWithTexture(heroTexture, cocos2d::CCRectMake(32*1, 0,
  32, 32));
  frame2 = CCSpriteFrame::frameWithTexture(heroTexture, cocos2d::CCRectMake(32*2, 0,
  32, 32));
  frame3 = CCSpriteFrame::frameWithTexture(heroTexture, cocos2d::CCRectMake(32*3, 0,
  32, 32));
    CCMutableArray<CCSpriteFrame *> *animFrames = new CCMutableArray<CCSpriteFrame*>(4);
    animFrames->addObject(frame0);
    animFrames->addObject(frame1);
    animFrames->addObject(frame2);
    animFrames->addObject(frame3);
    CCAnimation *animation = new CCAnimation();
    animation->initWithFrames(animFrames, 0.2f);
    animFrames->release();

    return animation;
}
```

有了NPC类，就可以在GameMap的初始化函数中读取object层中的所有对象，将它们创建成为NPC类的实例。我们新建了一个initObject方法，用于初始化对象层数据。CCTMXTIledMap类已经提供了相当方便的方法来访问对象层内的所有对象以及它们的属性。对象的属性存放在

CCStringToStringDictionary变量中，可以通过key来访问对应的value。

```
//初始化对象层
void GameMap::initObject()
{
    //初始化NPC对象
    npcDict = new CCMutableDictionary<int, NPC*>();
    //获取对象层
    CCTMXObjectGroup *group = this->objectGroupNamed("object");
    //获取对象层内的所有对象
    CCMutableArray<CCStringToStringDictionary *> *objects = group->getObjects();
    CCStringToStringDictionary *dict;
    CCMutableArray<CCStringToStringDictionary*>::CCMutableArrayIterator it;
    //遍历所有对象
    for( it = objects->begin(); it != objects->end(); it++)
    {
        dict = (*it);
        if(!dict)
            break;
        std::string key = "x";
        //获取x坐标
        int x = dict->objectForKey(key)->toInt();
        key = "y";
        //获取y坐标
        int y = dict->objectForKey(key)->toInt();
        CCPoint tileCoord = tileCoordForPosition(ccp(x, y));
        //计算唯一ID
        int index = tileCoord.x + tileCoord.y *this->getMapSize().width;
        key = "type";
        //获取对象类别
        CCString *type = dict->objectForKey(key);
        //如果类型是NPC对象
        if (type->m_sString == "npc"){
            NPC *npc = new NPC(dict, x, y);
            npcDict->setObject(npc, index);
        }
    }
}
```

最后，我们在Hero的checkCollision方法中加入NPC对象的碰撞检测：首先根据坐标计算出对应的index，然后根据index在npcDict中查询是否有对应的NPC存在，如果存在则与NPC进行交互。具体代码如下：

```
//从NPC字典中查询
int index = targetTileCoord.x + targetTileCoord.y *sGlobal->gameMap->getMapSize().width;
NPC *npc = sGlobal->gameMap->npcDict->objectForKey(index);
if (npc != NULL)
{
    actWithNPC();
    return kNPC;
}
```

为简单起见，仅在actWithNPC方法中调用gameLayer的showTip方法显示一条提示信息：

```
//与NPC交互
void Hero::actWithNPC()
{
    sGlobal->gameLayer->showTip("talking with npc", getPosition());
}
```

至此，我们大致完成了在游戏内添加NPC对象的目标。虽然与NPC的交互目前还比较简单，但已经为以后的扩展铺好了道路。

2. 传送点及地图切换

魔塔游戏与其他的RPG一样，都会有地图切换的需求。在魔塔中的体现形式就是上楼和下楼的传送点。首先在TiledMap中绘制传送点。打开Tiled编辑器，选中object层，选择工具栏中的“插入对象”按钮，在地图的(6, 1)处创建一个对象，编辑其属性如图11-15所示，其中“类型”项用于区分对象是否为传送点，heroTileCoordX和heroTileCoordY定义了勇士在目标地图中出现的坐标，targetMap定义了目标地图的层数。down_floor.png是用来显示的图片纹理，如图11-16所示。

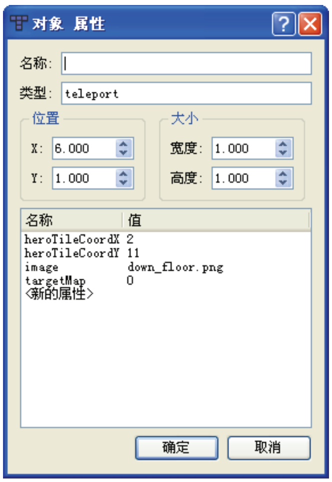


图11-15　传送点对象的属性配置页

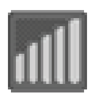


图11-16　下楼传送点的图片素材

下面我们来创建Teleport对象，用于记录上述配置信息，具体头文件如下：

```
#ifndef __TELEPORT_H__
#define __TELEPORT_H__

#include "MTGame.h"

using namespace cocos2d;
```

```
class Teleport : public CCObject
{
public:
    Teleport(CCStringToStringDictionary *dict, int x, int y);
    ~Teleport(void);
    //传送点所在位置
    CCPoint tileCoord;
    //传送到目标层后，勇士所在坐标
    CCPoint heroTileCoord;
    //目标地图的层数
    int targetMap;
    //唯一的ID
    int index;
    //图片纹理的文件路径
    CCString *imagePath;
    CCSprite *teleportSprite;
};
#endif
```

Teleport类的具体实现如下：

```
#include "Teleport.h"

Teleport::Teleport(CCStringToStringDictionary *dict, int x, int y)
{
    CCPoint position = ccp(x, y);
    //传送点所在的TileMap位置
    tileCoord = sGlobal->gameMap->tileCoordForPosition(ccp(x, y));
    //得出勇士在目标层的起始位置
    std::string key = "heroTileCoordX";
    int x1 = dict->objectForKey(key)->toInt();
    key = "heroTileCoordY";
    int y1 = dict->objectForKey(key)->toInt();
    heroTileCoord = ccp(x1, y1);
    //取得目标地图的层数
    key = "targetMap";
    targetMap = dict->objectForKey(key)->toInt();
    //获取image项
    key = "image";
    imagePath = dict->objectForKey(key);
    //创建用于显示Teleport的精灵
    teleportSprite = CCSprite::spriteWithFile(imagePath->m_sString.c_str());
    teleportSprite->setAnchorPoint(CCPointZero);
    teleportSprite->setPosition(position);
    sGlobal->gameLayer->addChild(teleportSprite, kZTeleport);
}

Teleport::~Teleport(void)
{
}
```

在GameMap的initObject方法中，遍历object层的所有对象时，判断对象类型如果是

teleport，则新创建一个Teleport对象，并存放到teleportDict中。

```
//如果类型是传送门
if (type->m_sString == "teleport"){
    Teleport *teleport = new Teleport(dict, x, y);
    teleportDict->setObject(teleport, index);
}
```

修改Hero类的checkCollision方法，添加传送点的碰撞检测逻辑：

```
//从Teleport字典中查询
Teleport *teleport = sGlobal->gameMap->teleportDict->objectForKey(index);
if (teleport != NULL)
{
    doTeleport(teleport);
    return kTeleport;
}
```

在doTeleport方法中，需要实现切换地图的逻辑。我们也许会犹豫不决：切换游戏地图的操作到底是切换GameScene呢，还是切换GameLayer？事实上两种方式都是可以的，只不过在效率上有一些差别：切换GameLayer操作实际上仅更新了GameMap和Hero对象，对ControlLayer实际并无影响；而切换GameScene会重新创建GameLayer以及ControlLayer。因此切换GameLayer的效率会高一些。所以我们这里打算使用切换GameLayer的方法来实现切换游戏地图的操作。由于GameLayer是依附于GameScene对象的，我们需要在GameScene中添加一些方法，用于销毁当前的GameLayer对象，再重新生成目标地图的GamLayer对象。这里新建了一个方法，即GameScene::switchMap，它的实现代码如下：

```
//切换游戏地图
void GameScene::switchMap()
{
    //创建一个遮罩层，用于地图切换时的显示淡入淡出效果
    CCLayerColor *fadeLayer = CCLayerColor::layerWithColor(ccc4(0, 0, 0, 0));
    fadeLayer->setAnchorPoint(CCPointZero);
    fadeLayer->setPosition(CCPointZero);
    addChild(fadeLayer, kFadeLayer, kFadeLayer);
    //执行淡入动画，结束后调用resetGameLayer方法
    CCAction *action = CCSequence::actions(
        CCFadeIn::actionWithDuration(0.5f),
        CCCallFunc::actionWithTarget(this,
        callfunc_selector(GameScene::resetGameLayer)),
        NULL);
    fadeLayer->runAction(action);
}
```

为了实现地图切换时的淡入淡出效果，新创建了一个CCLayerColor层，颜色为黑色，先让其执行CCFadeIn动作，视觉效果就是当前场景逐渐被黑色遮罩直至完全覆盖，实现了淡出效果。CCFadeIn执行结束后，会调用resetGameLayer方法，执行真正的切换逻辑。

```
//切换游戏地图之前
void GameScene::switchMap()
{
    //创建一个遮罩层，用于地图切换时的显示淡入淡出效果
    CCLayerColor *fadeLayer = CCLayerColor::layerWithColor(ccc4(0, 0, 0, 0));
    fadeLayer->setAnchorPoint(CCPointZero);
    fadeLayer->setPosition(CCPointZero);
    addChild(fadeLayer, kFadeLayer, kFadeLayer);
    //执行淡入动画，结束后调用resetGameLayer方法
    CCAction *action = CCSequence::actions(
        CCFadeIn::actionWithDuration(0.5f),
        CCCallFunc::actionWithTarget(this,
        callfunc_selector(GameScene::resetGameLayer)),
        NULL);
    fadeLayer->runAction(action);
}
```

resetGameLayer方法如下。开始时先删除当前的gameLayer对象，再用node方法创建新的gameLayer，最后让遮罩层执行CCFadeOut操作，并且注册了一个回调函数，用于移除遮罩层：

```
//切换游戏地图
void GameScene::resetGameLayer()
{
    //删除老的GameLayer
    this->removeChildByTag(kGameLayer, true);
    //创建新的GameLayer
    GameLayer *gamelayer = GameLayer::node();
    this->addChild(gamelayer, kGameLayer, kGameLayer);
    //遮罩层执行淡出效果，结束后，调用removeFadeLayer方法删除遮罩层
    CCAction *action = CCSequence::actions(
        CCFadeOut::actionWithDuration(0.5f),
        CCCallFunc::actionWithTarget(this,
        callfunc_selector(GameScene::removeFadeLayer)),
        NULL);
    this->getChildByTag(kFadeLayer)->runAction(action);
}
```

在removeFromFadeLayer中，仅需要删除遮罩层即可：

```
void GameScene::removeFadeLayer()
{
    this->removeChildByTag(kFadeLayer, true);
}
```

我们发现switchMap方法并没有向GameLayer传递任何数据，那么怎么通知GameLayer关于新地图的信息呢？为了避免来回传参的复杂性，我们简单地在Global类中新增了两个变量，用以标识目标地图的层数以及勇士的起始位置。

```
//目标地图的层数
int currentLevel;
//勇士出现的起始位置
```

```
CCPoint heroSpawnTileCoord;
```

随后修改GameLayer的init方法，让其根据这两个变量来动态读取地图和设置勇士位置。

```
//解析tmx地图
char temp[20];
sprintf(temp, "%d.tmx", sGlobal->currentLevel);
map = GameMap::gameMapWithTMXFile(temp);

//调用Hero类的静态方法创建实例
hero = Hero::heroWithinLayer();
//设置Hero的起始位置
hero->setPosition(map->positionForTileCoord(sGlobal->heroSpawnTileCoord));
```

由于修改了GameLayer的初始化方法，在GameScene的开始新游戏的方法playNewGame中需要对上面两个变量进行赋值：

```
//新游戏，当前地图层数为
sGlobal->currentLevel = 0;
//勇士出生位置
sGlobal->heroSpawnTileCoord = ccp(1, 11);
```

同样，在Hero类的doTeleport中先给这两个变量赋值，再调用GameScene的switchMap方法就可以轻松实现地图的切换了：

```
//传送点逻辑
void Hero::doTeleport(Teleport *teleport)
{
    //从传送点的属性中后去目标地图的层数
    sGlobal->currentLevel = teleport->targetMap;
    //获取勇士在新地图中的起始位置
    sGlobal->heroSpawnTileCoord = teleport->heroTileCoord;
    //开始切换游戏地图
    sGlobal->gameScene->switchMap();
}
```

下面将0.tmx复制一份命名为1.tmx，读者可自由发挥修改其中的元素。将第0层的传送点的targetMap设置为1，第1层的传送点targetMap设置为0，就可以控制勇士在第0层和第1层之间来回切换了。

11.2.4 小结

这一节中，我们向游戏中添加了丰富的元素：怪物、物品、门、NPC和传送点。下面来回顾一下主要知识点。

(1) 如何在TileMap中绘制块层。

(2) 如何在代码里动态更新图块ID，实现动画效果。

(3) 如何为怪物添加受打击的动画。

(4) 如何在TileMap中添加对象层。

(5) 如何在代码中解析对象属性。

(6) 如何实现切换游戏地图的功能。

11.3　总结

本章首先讨论了代码设计方面的问题，提供了一个良好的代码结构，可以很方便地扩展功能，然后又为游戏增添了大量元素，提高了游戏的可玩性。

本章以及前一章的游戏设计思想和代码是基于MT工作室的《魔域之城》游戏的，在此感谢MT工作室的慷慨奉献。当然，一个完整的魔塔游戏的内容远远大于这两章介绍的，因此还需要读者进一步研究、学习。

在最后一章里，我们会分析一下智能手机系统、手机游戏和cocos2d-x引擎的未来发展趋势，作为本书的结束。

第12章

未来展望

至此，我们的cocos2d-x之旅也要告一段落了。通过阅读本书，学习书中的例子，相信你已经对coocs2d-x这个能够跨iOS、Android和沃Phone平台的游戏引擎有了全面的了解。基于本书的案例，相信你一定能够开发出一个完美的游戏。但是除了书中介绍的知识外，你还需要更多地了解智能手机系统未来的发展，你还需要分析手机游戏未来的发展方向，当然你也需要知道coocs2d-x这个神奇的引擎未来会有哪些发展。

12.1 智能手机系统的发展趋势

手机系统的发展一直都牵动着手机行业。刚开始，只有大的手机厂商（如诺基亚和索爱等）才有自己的一套系统，后来，MTK解决方案出来之后，国内很多厂商都进入了手机行业，这样就有了更多的手机可供我们选择。

刚开始，我们用手机内支持的应用（如短信、闹钟和电话等）以及游戏（如《贪吃蛇》和《吃豆子》等）。而现在，智能手机已经比较普遍，智能手机除了具备通话功能外，还具备了PDA的大部分功能，特别是个人信息管理以及基于无线数据通信的浏览器、GPS和电子邮件功能。智能手机为用户提供了足够的屏幕尺寸和带宽，既方便随身携带，又为软件运行和内容服务提供了广阔的舞台，很多增值业务可以就此展开，如股票、新闻、天气、交通、商品、应用程序下载和音乐图片下载等。通过安装新的应用，我们可以让手机拥有更多的功能，如移动办公、手机邮箱、手机新闻和移动炒股等；通过安装新的游戏，我们可以让手机拥有更强的娱乐性，如《愤怒的小鸟》、《植物大战僵尸》、各类农场和魔塔游戏等。

那么，让我们来看看目前几大智能手机系统的发展趋势。

12.1.1 iOS的发展趋势

iOS是苹果的iPhone手机和iPad平板电脑使用的智能手机系统。

每一版iPhone和iPad的热卖，都会给iOS系统带来一波新的浪潮。但也正因为如此，如果未来iPhone或iPad不再受到追捧，那么iOS系统的市场占有率应该会随之下降。不过从目前来看，iOS的未来还是值得期待的，在作者编写本书之际iPhone 5的消息已经此起彼伏，看起来会继续大卖。

iOS目前只被iPhone手机和iPad平板电脑使用，很容易形成一荣俱荣一损俱损的状态。

但是iOS是几个智能手机系统中App Store做得最好的，这就给iOS带来了软件的支撑，目前苹果的App Store上的应用和游戏不计其数，而且由于App Store在整个产业链上做得比较好，能够给开发者带来收益，今后应该会有越来越多的应用和游戏进入其中，这也意味着在App Store上的竞争会越来越激烈。

12.1.2 Android的发展趋势

Android 是谷歌发布的一款开源的智能手机系统。Android系统最近是风光无限，HTC在这几年内连续发布了多款基于Android系统的手机，例如HTC的G1、G2、G3、G4、G7等；摩托罗拉也发布了基于Android系统的里程碑系列手机，其他很多手机巨头都是它的拥护者。

很多厂商基于Android定制属于自己的系统，中国移动的OPhone和中国联想的LEOS就是Android的定制版本。

Android系统的确给它的拥护者们带来了丰厚的回报，其中最为突出的就是摩托罗拉，在Android出来之前，摩托罗拉正在逐渐跌入谷底。摩托罗拉凭借着多款基于Android系统的里程碑系列手机强势反弹，已经一举走出困境，重新回到顶级手机厂商的竞争之列。

目前来看，Android可以说是：系统优异、机型众多、广受关注、厂商支持、免费开源、软件数量与日俱增，该系统未来的发展前景不可估量。

但是由于Android是开源的，基于Android的商城日益增多，包括谷歌的Android Market、亚马逊的Android商城和摩托罗拉的智件园等，这些商城的发展步伐不一，而且容易分散用户的精力，使得任何一个单独的Android商城都还达不到苹果公司的App Store的地位，所以很多开发者对Android的盈利能力抱有怀疑态度。

综上所述，Android系统是一个发展前景无限美好的系统，可只有Android的产业链也开始配套的时候，才能有更多开发者进入这个系统，才能有更多应用和软件进入这个系统，进入这个产业链。

12.1.3 沃Phone的发展趋势

进入3G时代，谷歌、苹果和微软等国际巨头纷纷推出智能手机系统，借助手机系统向移动互联网转型。谷歌发布了Android，苹果发布了iOS，微软发布了Windows Phone 7。

整个手机市场形成了一波新的Phone浪潮：苹果的iPhone，中国移动基于Android平台定制的OPhone，中国联想基于Android平台定制的乐Phone，中国联通基于Linux平台自主研发的沃Phone。

在IT发展的过程中，操作系统一直担当着非常重要的角色。Windows的成功成就了微软，而iOS的成功造就了苹果。操作系统是IT的基础，手机系统就是移动互联网的基础。有了基础才有可能把分散的力量组织起来。

沃Phone是由中国联通主导自主开发的，它是一个以Linux内核为基础的手机系统，具有自主、开放的特性，而且中国联通已经和众多国内外手机厂商合作推出了一批基于沃Phone的机器。因

此沃Phone今后的发展很大程度取决于中国联通的态度。不过沃Phone毕竟是国内第一个自主研发的智能手机系统，中国联通应该会扩大推广力度。但是要成为另一个iOS，还需要很多人的努力。

12.2　手机游戏的发展趋势

电脑游戏从单机游戏发展到网络游戏，网络游戏从按时间收费转变到按道具收费，甚至还出现了专门收取手续费的游戏。由于网络环境越来越好，电脑性能越来越强，开发电脑游戏逐渐形成了一个庞大的工程，一般大型的电脑游戏估计要几百人同时开发几年，才能做出一个网络游戏。这样庞大的工程体系还不适合手机。电脑上另一大分支的游戏就是网页游戏和社交游戏，这类游戏主要是通过网页进行操作，不需要用户长时间地操作电脑，而且可以直接连接社交网络，建立社交关系，是目前一个新的趋势。

目前，手机上已经出现了的一些MMORPG（Massive Multiplayer Online Role Playing Game，大型多人在线角色扮演类游戏）网络游戏，其特点是随着玩家的游戏时间变长，其操作的虚拟角色会有“成长”，游戏乐趣来源是“虚拟成长”带来的成就，角色成长后更强大，可以获得更强的装备、挑战更强的怪物、打败其他玩家。这种游戏追求黏性，希望玩家玩这个游戏的时间越长越好。另一方面，由于游戏时间长或者网络限制，所以游戏节奏比较慢，比如慢慢走地图迷宫和NPC（游戏内角色）对话时一页一页地翻看文字内容，同时游戏系统庞大复杂，可以让玩家游戏时间更久，延长产品的生命周期。

而笔者个人认为，手机游戏发展的主流方向一定不是像MMORPG那样需要占用长时间的网游。从苹果的App Store和谷歌的Android Market上的现有的游戏可以看出，绝大多数的游戏还是单机小游戏，这些游戏都具有以下几个特点：时间短、节奏快、靠操作过关而不是靠角色成长过关。

基于上述种种原因，笔者的个人预测是：手机游戏今后的主流会是“弱联网化、弱社区化”的单机游戏和“真实时间收益”的网络游戏。笔者认为现在的一些公司在手机上按照电脑上网络游戏的传统观念将手机网游做成“虚拟时间收益”的网络游戏，这一点是错误的。关于“弱联网化、弱社区化”、“真实时间收益”、“虚拟时间收益”这几个概念请看下文的分析。

在手机游戏方面，同样存在着单机游戏和网络游戏，下面分别看一下这两大类游戏。

12.2.1　手机单机游戏

单机游戏在国内存在一个比较大的问题，就是版权很难得到保证。做电脑单机游戏的公司不是改行做网络游戏就是已经关门了。在手机平台上版权问题稍微有点好转，因为苹果的App Store和谷歌的Android Market等商城在一定程度上会保护版权。正因为如此，现在已经存在一些只做单机游戏而过得很不错的厂商了，典型的就是智乐软件（GameLoft）。

大家仔细观察苹果的App Store和谷歌的Android Market等商城里卖的游戏，大多数还是单机游戏，让用户花一两美元买来玩，玩一两周后扔掉，手机游戏正在成为一种“快速消费品”。玩

家在游戏中获取的主要乐趣是操作的乐趣或者来自于童心的乐趣，而不在于传统网络游戏中“成就型、探索型、社交型、杀手型”的乐趣。这类单机游戏的节奏一般比较快、游戏系统简单、玩家比较容易上手，比网络游戏更适合在手机上进行短时间的娱乐。

不过单机游戏除了版权之外还存在一个比较大的问题，那就是游戏的生命周期太短，用户黏性差，游戏开发者很难获得长期的稳定收入。因此这些单机游戏需要通过一些简单的社交功能来吸引用户，黏住用户。这些简单的功能大致包括提供积分排行榜以及提供等级等，这个都是弱社区化、弱联网化的功能。

正因如此，笔者认为手机游戏在单机方面的发展趋势，应该就是这类的“弱联网化、弱社区化”的单机游戏。这种趋势在苹果的App Store和谷歌的Android Market等商城上已经非常明显。

12.2.2 手机网络游戏

关于网络游戏，我们先抛开手机网络游戏，来看看网页游戏市场，尤其是社交游戏。网页游戏和传统网络游戏的一个很大区别就是玩法不同。

传统网络游戏主要是靠“虚拟世界时间”来实现角色成长，角色要想积累虚拟货币、经验值和等级，就需要长时间地泡在游戏中，一般来说玩家在虚拟世界里待得时间越长，等级就越高。玩家一旦关掉游戏，则虚拟货币、经验值和等级的成长就会暂停；而网页游戏主要则是靠“真实世界时间”来实现成长的，在“真实世界时间”下即使玩家把游戏关掉，虚拟游戏世界里仍然会按照真实时间来增加虚拟货币、经验值和等级，在虚拟货币、经验值和等级积累到一个上限后需要玩家进行操作。

因此传统网络游戏需要玩家在游戏过程中专心地操作游戏角色，在游戏过程中比较难分心去做其他事，更不能边游戏边工作，需要每天集中玩几小时。而网页游戏，不论是农场、餐厅还是三国，典型玩法都是把这几小时的时间平均到全天中，每隔半小时或一小时需要关注一下虚拟游戏世界、操作一下，不过每次操作的时间都可以很短，几分钟甚至几十秒就可以了。当按照“虚拟时间收益网游”和“真实时间收益网游”来区分后，我们更能看清楚网络游戏了。

在网页游戏里，我们经常能够看到类似下面这样的提示：

(1) 你的种子30分钟后成熟，成熟10分钟后存在被偷的风险，建议提前准备收割；

(2) 敌方部队在10分钟后到达；

(3) 20名骑兵在80分钟后生产完毕；

(4) 玩家A因为刚才偷菜被发现，因此被城市警察抓住，30分钟后释放。

即使加入付费道具也很简单，对于“虚拟时间收益网游”就是使玩家在花费同样单位的虚拟世界时间，能得到更高的收益；而对于“真实时间收益网游”就是使玩家在花费同样单位的真实世界时间，能得到更高的收益。

现在我们回到手机网络游戏的话题上，如果玩家在手机上玩“虚拟时间收益网游”，不论在iOS、Android还是在沃Phone上，都会出现以下问题：

(1) 长时间操作游戏会耗电；

(2) 由于需要专心操作角色，因此对游戏画质、音质都有更高的要求；

(3) 玩家沟通不便，打字聊天沟通困难，输入文字不便（尤其是中文）；

(4) 长时间操作游戏，容易引起身体不适。

如果想花费大段时间玩虚拟时间收益游戏，那么一定会选择电脑上的网络游戏，而不会提着手臂、耷拉着脑袋、冒着颈椎受损的危险在手机上玩。

正因如此，笔者认为手机上的网络游戏，最后会呈现出以"真实时间收益网游"为主流的趋势。就类似于发短信打电话一样，操作频度很高，但每次操作时长都很短。这样就不会存在上述提到的几个问题。当然"真实时间收益网游"不等于"社交游戏"，"真实时间收益网游"同样可以是策略类游戏、养成类游戏或者RPG类游戏。只要设计成随着真实时间流逝能有虚拟的成长，都可以落在这个范围内。

总结可以得出：手机游戏今后的主流会是 "弱联网化、弱社区化"的单机游戏和"真实时间收益"的网络游戏。其实这两类游戏的本质都是如何让玩家用碎片时间玩游戏并且获得成长和乐趣。

从电脑游戏的发展史可以看出：谁先发掘出玩家的问题，并且有创意地解决了，谁就能赚到第一桶金。例如，满地图任务、装备绑定、副本、跨服战场、等级上限、无尽装备、装备砸宝石、概率性升级、公会战、攻城战、跑环任务，这些概念的创造者基本都赚到了。

12.3　cocos2d-x 引擎的发展趋势

通过与cocos2d-iphone引擎、cocos2d-x的作者以及其他专家的交流和分析，笔者认为cocos2d-x引擎的发展主要有以下几种可能性。

第一种可能是，紧跟cocos2d-iphone引擎的功能，继续完善跨平台性。目前cocos2d-x引擎已经能够支持苹果iOS、谷歌Android、微软Windows XP、微软Windows 7、联通沃Phone、联想乐Phone、Meego、Marmalade（原AirPlay SDK），将来继续追加更多的手机平台，比如升级年久失修的Bada版本移植，或者增加WebOS、BlackBerry版本等。当然这其中大家最期待的就是能否移植到微软Windows Phone 7上了。如果能跨iOS、Android和Windows Phone 7三个主流平台，那么就谁都无法阻挡cocos2d-x引擎的飞速发展了。

第二种可能是，在C++语言的基础上，增加多种编程语言的绑定。现在我们可以看到官方已经推出了Lua语言的绑定，未来不排除加上Java、JavaScript、C#和Python等编程语言的可能性。这种理论在技术上是完全可行的，Unity3D就是这样的技术方案，目前还不确认cocos2d-x引擎是否会走这条路。

第三种可能是，移植cocos3d引擎，成为一个集2D和3D功能于一身的大引擎。如果是这样，那么打算用cocos2d-x引擎做2.5D游戏的开发者就有福了。

第四种可能是，推出动作编辑器，目前cocos2d-iphone引擎社区里已有两个开源的编辑器在快速发展，这两个开源的编辑器分别是CocosBuilder和CocoShop。Ricardo Quesada多次在论坛里强调新开发的编辑器输出格式为plist格式，这是一种在cocos2d-iphone引擎和cocos2d-x引擎上都能兼容的格式。

第五种可能是，往Web方向发展，HTML5是未来的趋势，Unity3D也推出了浏览器插件以支持Web游戏的开发，同时我们也看到了cocos2d-javascript分支的发展。未来cocos2d-x引擎是否有可能演变WebGL的技术路线呢？cocos2d-iphone论坛里就cocos2d 2.0路线中脚本选择Lua还是JavaScript争论不休，Ricardo Quesada表示他个人比较倾向于选择JavaScript向Web方向发展，Walzer也提到了用Google V8引擎绑JavaScript的可能性。

针对上面的每一种可能性，大家都能说出很多种理由来支持，也能说出很多种理由来反对。Ricardo Quesada和Walzer都没有对未来路线明确表态，Walzer表示cocos2d-x引擎会和游戏开发者一同成长，发展为一个受开发者欢迎、切实解决开发困难、减少游戏开发工作量的开源引擎。

cocos2d-x手机游戏开发

跨iOS、Android和沃Phone平台

纵观移动互联网时代各大移动平台的发展，游戏无疑是大众娱乐的首选。由于各个平台的差异性，游戏开发的难度和成本逐渐增加，“跨平台”一词也成为了移动游戏开发者最关注的焦点，还等什么呢？如果你想一次编码，多平台运行，cocos2d-x正是你需要的，而本书则由浅入深地介绍了cocos2d-x游戏开发的全过程，理论与实践相结合，是移动游戏开发者不可多得的佳作。强烈推荐！

——杨丰盛（yarin），More-Top团队创始人，畅销IT图书作家

在移动客户端发展愈演愈烈的今天，移动开发技术已经成为了主流之一。我们看到很多移动开发团队取得了成功，而成功并不是一蹴而就的，我们需要的是若干年的积累，最后形成质的飞跃。《cocos2d-x手机游戏开发》这本书，也许就是您走向明天成功的开始，cocos2d-x，可能就是您走向成功的基石。

——赵磊，51CTO副总编

喜欢游戏，不是因为某平台的各种一夜暴富的传说，而是儿时心中的梦想。轻盈、纯洁、开放、坦诚地面对每一个朋友，cocos2d-x无疑是最合适的。感谢本书的作者给我们揭开cocos2d-x神秘的面纱，带我们走进这个世界。在移动互联网和移动应用火热的今天，我们需要这样一个向导，为我们打开通往这个世界的传送门，去这个全新的领域追寻、探索。不要错过这个机会，不要让自己的梦想冷却，深埋心底。

——王明杨，永杨安风（LBE安全大师）CEO

目前，在苹果App Store和谷歌Android Market的排行榜内，收入较高的绝大部分都是游戏。市场和用户选择了游戏，所以我们也选择了游戏。本书详尽、全面、深入地介绍了如何使用游戏引擎开发优质游戏以及如何快速集成游戏社交平台、手机广告平台、虚拟物品及流量互换的推广墙服务，从而快速开发、快速扩大用户规模和获取积极的游戏收入。本书的出版恰逢其时，是手机游戏开发者必备的佳作，强烈推荐！

——刘琦，Wiyun Inc.（微云）COO



徐松林 智能手机平台的先驱者，曾就职于国内知名的企业级平台软件公司，主攻复杂系统的架构设计，在此期间积累了有关大型软件架构设计的丰富经验。2010年5月成立MT工作室，设计并开发用于Android平台的游戏引擎（MT引擎），并基于此引擎开发出多款市场反馈较好的游戏。一直专注于研究跨平台的游戏引擎，尤其是跨平台引擎的架构及设计原理。



黄　猛 专注于移动平台游戏开发，对Android以及iOS系统的编程有深入研究。自Android发布以后，一直关注移动平台的发展，从2008年底开始从事Android底层应用以及2D游戏的研发，有丰富的实战经验，是《魔塔之拯救公主》、《超级DJ》和《魔域之城》等经典游戏的主力开发者。

图灵社区：www.ituring.com.cn
反馈／投稿／推荐信箱：contact@turingbook.com
热线：(010)51095186转604

分类建议 计算机／移动开发／游戏开发

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-26766-5

ISBN 978-7-115-26766-5
定价：39.00元